An Energy-Efficient Periodic Resource Model for Cyber-physical Systems

by
Suzanne Elashri

A thesis submitted to the
School of Graduate and Postdoctoral Studies in partial
fulfillment of the requirements for the degree of

Master of Applied Science in Electrical, Computer and Software Engineering.

Faculty of Engineering and Applied Science

Department of Electrical, Computer and Software Engineering

University of Ontario Institute of Technology (Ontario Tech University)

Oshawa, Ontario, Canada

August, 2021

Thesis Examination Information

Submitted by: **Suzanne Elashri**

Master of Applied Science in **Electrical, Computer and Software Engineering**

Thesis Title: An Energy-Efficient Periodic Resource Model for Cyber-physical Systems.

An oral defence of this thesis took place on July 29,2021 in front of the following examining committee:

Examining Committee:

Chair of Examining Committee	Ying Wang
Research Supervisor	Akramul Azim
Examining Committee Member	Shahryar Rahnamayan
Thesis Examiner	Mikael Eklund

The above committee determined that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate during an oral examination. A signed copy of the Certificate of Approval is available from the school of Graduate and Postdoctoral studies.

ABSTRACT

Cyber-Physical Systems (CPS), especially real-time systems, are subject to time constraints that should be met for the applications to run properly. The schedulability of workloads can be analyzed by determining the supply and demand bound functions (*sbf* & *dbf*) when the minimum resource availability (*sbf*) can satisfy the maximum possible resource demand (*dbf*) of the workload under a specific scheduling algorithm during a given time interval.
Typical real-time systems are insufficient resource reservation by over-provisioning resources when resource supply is always higher than the workload demand. Energy efficiency considerations are required for resource reservations in real-time systems when calculating the optimum processor speed to ensure that the supply of resources is no less than the workload demand during any time intervals. Therefore, we explore an energy-efficient Dynamic Speed Scaling technique for CPS to efficiently estimate resource reservation and efficiently reduce energy consumption.

Keywords: Cyber-physical systems; Periodic resource model; Energy consumption; Delay-tolerant tasks.

AUTHOR'S DECLARATION

I hereby declare that this thesis consists of original work of which I have authored. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize University of Ontario Institute of Technology (Ontario Tech University) to lend this thesis to other institutions or individuals for scholarly research. I further authorize University of Ontario Institute of Technology (Ontario Tech University) to reproduce this thesis by photocopying or other means, in total or in part, at the request of other institutions or individuals for scholarly research. I understand that my thesis will be made electronically available to the public.

Suzanne Elashri

STATEMENT OF CONTRIBUTIONS

I hereby certify that I am the sole author of this thesis and the sole source of the creative works and inventive knowledge described in this thesis.

The contributions of my work are listed as follows:
We propose an energy-efficient periodic resource model using Dynamic Speed Scaling *DSS* for cyber-physical systems.
We prevent over-provisioning the resources for delay-tolerant real-time systems by permitting the task workload to exceed the resource supply within a transient bounded delay. Moreover, We propose an online and offline offloading algorithm for energy-efficient delay-tolerant real-time tasks.
For hard real-time systems, we propose an efficient energy-aware scheduling optimization technique under *DSS* for task and component levels (OPT-TDSS, OPT-CDSS) to reduce over-provisioning resources while still guaranteeing timing constraints (mainly deadlines).
The contributions have already been published as:

- Suzanne Elashri and Akramul Azim. "An Energy-Efficient Periodic Resource Model for Bounded Delay-Tolerant Real-Time Systems." IEEE Access (2021) [1].

- Suzanne Elashri and Akramul Azim. "Energy-efficient offloading of real-time tasks using cloud computing." Cluster Computing (2020): 1-16 [2].

Another work has been accepted at the premier international conference on embedded software (EMSOFT) 2021:

- Suzanne Elashri and Akramul Azim. "Work-in-Progress: An Energy-Aware Optimization Model for Real-Time Systems Analysis and Design." International Conference in Embedded Software (EMSOFT) 2021.

ACKNOWLEDGEMENTS

I would first like to express my gratitude and appreciation to my supervisor Dr. Akramul Azim, Assistant professor, Ontario Tech University, for his valuable comments, observations, and engagement throughout the master's thesis learning process. He provided me with meaningful advice, encouragement, and inspiration while I worked on this thesis. Dr. Akramul Azim always gives me the needed time, and he makes the virtual learning process (during the pandemic) very easy on me. He continually enabled me to work in my own best schedule and interests while also pointing me on the proper path when necessary. Thank you for believing in me and assisting me with every part of my thesis.

Furthermore, I would like to thank all of my RTEMSOFT research group members who have supported me throughout the process, both by helpful criticism sharing thoughts to improve my work. Many thanks to the Software System Research Lab members who contributed to the friendly atmosphere in my workplace. I appreciate the support of all my friends, colleagues, and relatives who always motivated and encouraged me through this journey.

Finally, I want to express my profound gratitude to my family, especially my husband, Mohamed Abdelsalam, who gives me mental support whenever I needed it. I'm very grateful for his unconditional support, continuous encouragement, and the invaluable sacrifices made throughout my study. Also, I'm very thankful to my mother-in-law Nahid Keshta for her support and for taking care of my kids and me and making a calm environment throughout my study. I am grateful to my beloved mom Hemmat Elmasry, and dad Mahmoud Elashry who have prayed for my success and for always believing in me. I'm very thankful to my brother-in-law Ahmed Abdelsalam, my sister Mayada Elashry, and my brother Ahmed Elashry

for their continuous support and encouragement. This accomplishment would not have been possible without them. Thank you.

Table of Contents

List of Figures

List of Tables

List of Abbreviations and Symbols

CPS	Cyber-Physical System
RTS	Real-Time System
HRS	Hard Real-Time System
DRS	Delay-tolerant Real-Time System
WRT	Weakly hard Real-Time
COD	Least Computation Offloading Decision
SBF	Supply Bound Function
DBF	Demand Bound Function
PDBF	Power Demand Bound Function
EDF	Earliest Deadline First
DVFS	Dynamic Voltage Frequency Scaling
DVS	Dynamic Voltage Scaling
DSS	Dynamic Speed Scaling
WCET	Worst-Case Execution Time
LCM	Least Common Multiple

Different symbols used in this thesis can be listed as:

- C is a component, where each C is modelled as $C = {}^{1}W, A^{o}$.
  - W is the workload that has a set of independent tasks $W = f\tau_1, \tau_2, \ldots, \tau_n g$.
  - A is the scheduling algorithm that defined as earliest deadline first (EDF).
- $\tau = f\tau_1, \tau_2, \ldots, \tau_n g$, is the set of independent periodic tasks.
- $\tau_i = {}^{1}p_i, e_i^{o}$ is characterized as a task which is defined by as $\tau_i \; 2 \; \tau$ where
  - p_i is the time period of the task τ_i.
  - e_i represents the execution time of task τ_i.
- $R^{1}\Pi, \Theta^{o}$ is the periodic resource model, where is the duration $\Pi > 0$, and is the periodic time of allocation (< 0).
- C is the loading capacitance.
- f is the clock frequency.
- V is the operating voltage.
- I_{sc} is the short circuit current.
- I_{leak} is the leakage current.
- s is the processor speed.
- s_{min} is the minimum processor speed level and is equal to 0.
- s_{max} is the maximum processor speed level and is equal to 1.
- $m^{1}s^{o}$ different processor operating modes.
- E_t is the energy function.
- t_o,t_r are the points of interest for identifying the duration of delay.

- D is the actual delay.
  - t_o is the overload point.
  - t_r is the recovery point.
- σ is the worst-case tolerable delay.

Chapter 1

Introduction

In recent years, we observe a significant development of several software programs that can perform complex computations. In addition, the recent development of capabilities of real-time systems has led to a spectacular increase in their power consumption. Cyber-Physical Systems (CPS) are embedded systems controlled and modeled by a computer-based algorithm; they are firmly integrated with computation and communication. Nowadays, Cyber-physical systems are becoming increasingly common and capable of performing very complex computation, especially With the rapid development of the Internet of Things (IoT). For instance, the complexity of applications running on smart devices resulted in a shortening of the battery life. While the quantity and complexity of these applications are increasing daily, the capacity of lithium-ion batteries, which are the source of power in smart devices, is slowly improving [3]. Moreover, the amount of heat dissipation is directly proportional to power consumption. Therefore high rates of power consumption result in elevated amounts of heat dissipation. Heat dissipation not only deteriorates the performance of computers and embedded systems but also reduces their durability. For instance, a 15 C increase in temperature from the tolerable temperature can reduce the life of electronic devices by half [4]. These facts motivate us to find an efficient solution to decrease the power consumption of Cyber-physical systems.

Cyber-physical systems consist of different real-time systems (RTS) and are classified as hard real-time systems (HRS) and delay-tolerant real-time systems (DRS). For HRS, any violations of deadlines may cause injuries, physical damage, or system failure [5], the deadline of a task is considered as a specific time when a task should finish its execution. The majority of automotive systems, avionics, security, nuclear systems, and any system with safety-critical applications are considered hard real-time systems. While on the other hand, DRS can handle missing the deadlines without having catastrophic consequences [6], but the system's performance might be degraded.

1.1 Periodic Resource Model for Cyber-Physical Systems

Delay-tolerant real-time systems can tolerate the occurrence of delays (i.e., when the application requires more resources than available) within a bounded time interval. For instance, multimedia programs with strict timing limitations on processing video frames can still function properly even if the system is unable to process all frames within the deadline; there would be only jittering or a loss of video quality.

Another example of delay-tolerable systems is feedback control systems [7] [8] that are robust to small delays in the feedback loop [9]. Control systems have been used in many applications ranging from a small-scale such as a thermostat used to control a domestic boiler in a small residential building, to large-scale such as industrial systems used to control processes or machines [10].

1.1.1 Supply and demand bound functions

Cyber-physical applications require sufficient resources for computation and communication to function, probably by meeting their timing requirements. The system should provide enough resources for the application to meet the deadlines and examine the schedulability of workloads using supply and demand bound

functions. The higher bound on the resources required by the application is determined by the demand bound function, while the lower limit on the resources given to the tasks is determined by the supply bound function.

To analyze the feasibility of tasks inside a component for a given periodic resource model, [11] introduced the feasibility analysis of *sbf* and *dbf* under the consideration that the processor always runs at full speed level, which consumes more power than required. Therefore, we introduce a new feasibility analysis, the Power-aware demand bound function (*Pdbf*), that takes into account the processor speed. Reducing the system's processor speed at a certain level leads to providing fewer resources than the system required because the more processor speed reduction we can have, the higher workload demand we get. Thus, the application might experience delay for bounded time interval, and the schedulability of real-time systems (i.e., meeting the task deadline) at this point is not guaranteed [12]. Therefore, it is crucial to determine an efficient Dynamic Speed Scaling *DSS* algorithm that minimizes energy consumption while satisfying the system's schedulability.

1.1.2 Schedulability test for hard real-time systems

Schedulability tests exist for a given set of applications to determine whether the application receives the desired resources or not. For instance, the schedulability test, based on supply and demand bound functions, for systems that cannot tolerate any delay such as hard real-time systems, show that insufficient resources are available for a given time interval by having the supply bound function (sbf) falls below the demand bound function (dbf).

Therefore, to avoid having a system delay, demand bound function should remain below the supply bound function to achieve the schedulability of the system. Because otherwise, the schedulability test will indicate that insufficient resource exists for a bounded time interval.

1.1.3 Schedulability test for delay-tolerant real-time systems

Delay-tolerant systems can tolerate the occurrence of overloads/delays; delays happen when the application requires more resources than available within a bounded time interval. For instance, if the system can tolerate delaying the tasks' execution time, the schedulability test will accept a system whose workload demand exceeds the resource supply. Using tolerable delay systems to reduce resource over-provisioning effectively is a promising technique for lowering energy consumption. The schedulability test based on *sbf* and *dbf* guarantees the application's efficiency (schedulability).

Delay may be temporary/transient or permanent. A transient overload has abounded time delay, which can occur due to long task execution times or the arrival of asynchronous events at the same time. For example, the feedback control model experience some delay in the feedback loop [13] which is categorized as weakly hard real-time systems. Other applications that can tolerate a transient delay are soft real-time applications such as audio or video-on-demand applications. A permanent delay has an unbounded time interval, and having a system with permanent delay means the system is poorly designed and unschedulable.

1.2 Energy Efficiency for CPS

1.2.1 Dynamic Voltage Frequency Scaling

Several power management techniques have been developed and studied to minimize the power consumption of computers and embedded systems. Dynamic Voltage Frequency Scaling (DVFS) is a promising power management technique that has been employed in real-time systems to improve power utilization by reducing the dynamic power consumption of a processor through adjusting either the supply voltage, processor speed, or clock frequency. Many techniques [14],[15], [16] have been suggested based on DVFS; these existing techniques are pessimistic in

resource reservation because they assume the resources to be always available. In contrast, the availability of the resources depends on the resource model that may be periodic. However, aggressive reduction of the clock frequency of the processor may reduce the system performance. Therefore, achieving energy-efficient computation for real-time systems without impacting the application's performance remains a challenge.

1.2.2 Offloading approach for delay-tolerant RTS

The schedulability analysis is feasible when the delay occurs within the acceptable bounded delay range. However, in some cases, the actual delay exceeds the maximum tolerable delay of delay-tolerant systems, and thus, the schedulability of the application at this point is not guaranteed. Therefore, to address such an issue, we expand the limited capabilities of local devices by introducing an energy-efficient computation offloading decision (COD) approach that improves mobile device energy efficiency. Even though battery technology has improved over time, it has not been able to keep up with these mobile systems' fast increase in energy consumption. As a result, offloading is utilized to save energy by transferring computing processes that consume a large amount of energy to more energy-efficient devices. we propose a promising offloading mechanism for deciding when and where a task can be offloaded (based on supply and demand bound functions) for delay-tolerant RTS and assume that hard real-time tasks always execute locally to avoid any non-deterministic delays.

We propose offline and online COD algorithms. The offline algorithm is based on supply and demand bound function while the online algorithm is based on the processor utilization, and both techniques achieve significant energy savings for a local device by guaranteeing the schedulability of delay-tolerant CPS. Timing requirements are essential in which the results might be unacceptable if the delay exceeds the acceptable bounded delay range. The offline offloading algorithm happened at the cloudlet when the local device experience bounded delay. Cloudlet is a form of local cloud, also defined as fog or edge node. We then propose the online algorithm for unbounded tolerable delay systems and mapping the delay

to quality of service (QoS). The online algorithm uses an efficient offloading decision based on the power consumption of the application and the CPU utilization. In the experimental analysis, monitoring the local computational device's power consumption, execution time, and CPU utilization provides us an overview of how the system behaves.

Results of the proposed online algorithm identify that power consumption and execution time of the running tasks are highly coupled (i.e., the application's power consumption increased by increasing the execution time). However, power consumption and resource utilization are not necessarily linearly scale. This encouraging fact motivates us to consider the power consumption and the processor utilization while proposing our online algorithm. Thus, for a particular task, the information on its power consumption and processor utilization is sufficient for the offloading decision making, whether to extend the resources by executing the task remotely or perform it locally.

1.3 Motivation

The state-of-the-art research on delay-tolerant cyber-physical systems has primarily focused on reducing heavy over-provisioning of resources without comprising the schedulability of the tasks. However, to the best of our knowledge, energy consumption is still not explored while guaranteeing the timing requirements for delay-tolerant systems. Indeed, energy is a critical issue for CPS due to many reasons, a) the extensive computing in safety-critical systems such as autonomous vehicles that consist of 100 electronic control units require a massive power supply to maintain the functionality of those systems, b) many CPS are battery-operated, and lowering the energy usage might result in significant energy gain by extending gadget life and reducing maintenance expenses, and c) chip temperature becomes a key concern since the power density of advanced electronic circuits has increased dramatically, potentially affecting system dependability and timing accuracy [17]. As a result of all these facts, energy conservation is critical, and finding efficient energy-aware techniques for CPS has become an important research topic.

1.4 Objectives

For a given workload specifications, find energy-efficient resource utilization techniques while guaranteeing the deadline requirements of each task. This can be achieved by:

1. Slowing down the processor speed level as much as possible to close the gap between supply and demand bound functions and thus, efficiently estimates the resource reservation to achieve the maximum power savings in hard and delay-tolerant real-time cyber-physical systems.

2. Extending the resource availability for tasks that can tolerate delays to enhance their performance in terms of execution/completion time and power consumption by utilizing cloud/cloudlets resources.

3. Formulating the resource allocation as an optimization problem.

1.5 Contributions

In summary, the main contributions of this thesis are:

1. We proposed an energy-efficient Dynamic Speed Scaling (DSS) for periodic resource model to reduce energy consumption and maintaining the scheduling of individual task.

2. We proposed a computational offloading decision COD technique for tolerable delay tasks that can execute on cloudlets and/or cloud.

3. We proposed an efficient energy-aware optimization technique for task and component levels (OPT-TDSS, OPT-CDSS) using supply and demand bound functions to estimate the resource requirements efficiently.

1.6 Organization of the thesis

We organize the thesis into five chapters. In Chapter 2, we discuss different energy-efficient techniques in the related works and basic terminologies that will help to

follow the remainder of our thesis work. Moreover, it contains the recent design energy management techniques, task scheduling under the supply and demand bound functions approach, and different energy-based offloading policies.

In Chapter 3, we present the proposed approaches of the thesis that has different sections for better understanding. We define the system model, including the schedulability analysis and power models for a real-time scheduling framework. After that, we present our methodology of achieving an efficient *DSS* level that can tolerate a transient bounded overload to preserve requirements. Moreover, we demonstrate the workflow of our proposed design to clarify each used step. Following an illustrative example to show how an efficient *DSS* level can reduce energy consumption while guaranteeing the schedulability of a bounded delay-tolerant real-time system. We also present our offloading methodology using two approaches, offline and online algorithms.

In Chapter 4, we present the experimental analysis of the thesis that has different sections for understanding the advantages of our proposed approaches. We evaluate the performance of our proposed methods and show the feasibility of integrating cloud computing to achieve considerable power savings. The economic results of using a cloud platform are also described in this chapter Moreover, we present the results of our simulations using Matlab and AMPL/KNITRO solver to solve the optimization problem and show the feasibility of having an appropriate *DSS* level over the energy savings in real-time control system.

Finally, Chapter 5 concludes the thesis and points out possible future research directions in the context of reducing resources over-provisioning to reduce the energy consumption while guaranteeing the schedulability of real-time embedded systems.

Chapter 2

Literature review

2.1 Introduction

Reducing the energy consumption of cyber-physical systems has been a hot research topic because of battery life limitations. CPS should consider the applications' timing requirements to deliver the proper results without affecting their performance. Therefore, researchers have been concerned with addressing the tradeoff between power and performance of cyber-physical systems. This tradeoff has emerged as one of the most important criteria for assessing modern computer systems because of the demands for mobility and reliability; energy efficiency has become a top priority for real-time systems. Typical real-time systems are inefficient in resource reservations and consume more power than required because they consider the resources always available. There has been a lot of work on energy-efficient CPS in the literature. Various energy management techniques have been proposed based on two most commonly used methods, Dynamic voltage Frequency scaling (DVFS) [18], [19], [20], which reduces power consumption by scaling both the voltage and operating frequency of a processor, and dynamic power management (DPM) [21], [22], which aims to reduce the power consumption of devices such as memory modules and other I/O devices. Although this work focuses solely on DVS, many researchers have proposed hybrid power management approaches that use DVS and DPM. Although several studies have been published on energy management techniques, the vast majority focused on DVFS

approaches that didn't consider real-time constraints for delay-tolerant real-time systems. Therefore, in this thesis, we focus on reducing resource over-provisioning for hard real-time systems and preventing over-provisioning for delay-tolerant real-time systems to reduce energy consumption while still guaranteeing real-time systems' timing requirements.

2.2 Real-time scheduling

2.2.1 Periodic resource and task model

An application can be modeled as a workload that has a set of recurrent tasks. Each task, in general, can be characterized by: arrival time, worst-case execution time (WCET), period, and deadline. The task-sets can follow various models depends on the application's scope [23]. For instance, the task model could be:
Periodic: where the tasks periodically occur after a specific time interval.
Aperiodic: where the tasks have irregular arrival times and unpredictable deadlines.
Frame-based: where all the tasks should finish execution within the single recurring frame.
Sporadic: in which the duration of work determines the minimum time between jobs arriving.

Also, tasks can be categorized as dependent or independent. Dependent tasks execution time depends on the completion of the prior task, where there are no restrictions for the independent task.
This thesis considers the periodic resource and workload models that execute a set of independent tasks on a single processor (no threading). Each task is given by a time period and execution time, and we assume the deadline of the task to be equal to the time period. The hyperperiod of all tasks' periods forms the cycle at which the system repeats its behavior.

2.2.2 Scheduling approach

2.2.2.1 Single core scheduling

scheduling tasks is the process of assigning them to resources while guaranteeing that various constraints (i.e., deadlines) are met.

This area involves two main subject matters, proposing methods to schedule various task sets and determining schedulability tests to guarantee that the task-set requirements are satisfied under a specific scheduling algorithm.

The schedule can be classified as a static or dynamic schedule. In static scheduling, the tasks' information is known, and the scheduling decision is taken at compile-time, such as the rate monotonic (RM) scheduling algorithm, where the task's priority is determined by its period. In dynamic scheduling, the scheduling decision is taken while the task is being executed, such as the earliest deadline first (EDF), where the task's priority is determined based on their deadlines. We use the scheduling algorithm EDF in our work which is considered to be the optimal dynamic scheduling algorithm on single-core processors [24].

2.2.2.2 Multi-core scheduling

Real-time task scheduling on multicore has become a top priority for the real-time scheduling domain as embedded systems advance into the multicore area. Partitioned scheduling and global scheduling are two of the most common approaches to this problem that have been proposed in the literature [25].

Partitioned scheduling is a static scheduling mechanism for multicore CPUs where tasks are assigned to cores during compilation time. The given tasks are then scheduled individually by each core [23]. The main advantage of partitioned scheduling is that each assigned task-set under individual core can be scheduled via existing single-core algorithms. Partitioned scheduling is considered to be a good option for the static workload. Global scheduling is the best option for varying workloads, and to ensure workload balance, tasks can also migrate between available processors at runtime. Any task migration typically adds extra overhead, and according to dynamic changes in the task environment, the system may experience unexpected behavior. These migrations not only slow down the scheduler

but also make designing schedulability tests more complex. As a result, in global scheduler schedulability testing, the migration overhead factor is assumed to be constant or negligible [23].

2.2.2.3 Schedulability test under supply and demand bound functions

The schedulability test in this thesis is based on supply and demand bound functions for CPS. In hard RTS, the supply bound function cannot be less than the demand bound function to guarantee the timing constraints. Therefore, to avoid having a system delay, the demand bound function should remain below the supply bound function $dbf_W(A, t)$ $sbf_R(t)$ to achieve the schedulability of the system. The supply bound function obtained from [11] described as the minimum supply resource needed by the system during a time period t.

$$sbf_R(t) = \begin{cases} t & (k-1)(\Pi-\Theta) + \max(t-(k-1)\Pi-\Theta, 0), \quad \text{if } t \geq [k_1, k_2], \\ (k-1)\Theta & \text{otherwise,} \end{cases} \tag{2.1}$$

where $k_1 = (k-1)\Pi - \Theta$, $k_2 = (k-1)\Pi - 2\Theta$, and
$k = \max(d(t) - \Pi - (\Pi - \Theta), 0)$.

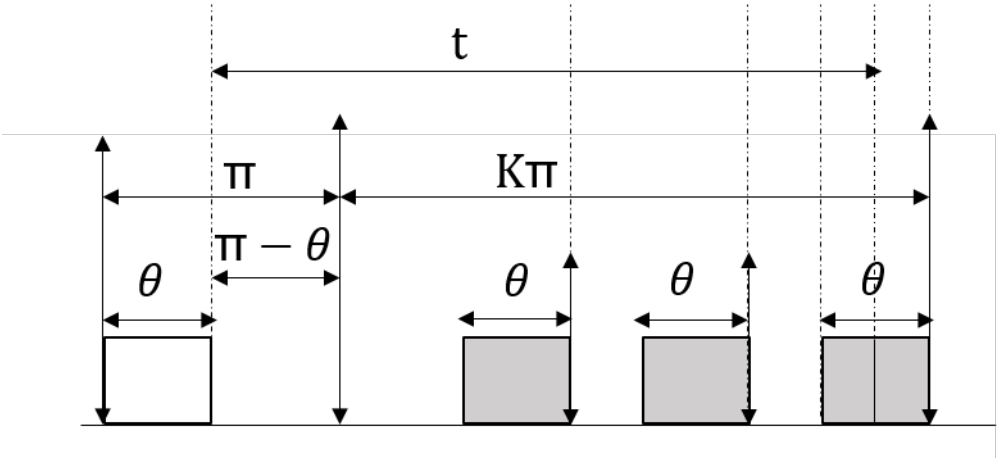


FIGURE 2.1: *sbf* of a periodic resource model $R(\Pi, \Theta)$ for $K = 3$

Figure 2.1 shows how we define the *sbf* (Equation 2.1) for $k = 3$.
$R(\Pi, \Theta)$ is the periodic resource model, where is the duration $\Pi > 0$, and is the periodic time of allocation ($0 <$). The periodic resource model's periodic capacity is defined as (•), and by analyzing supply and demand

bound functions using equation 2.3, 2.1, the optimum processor speed level will be determined for hard and delay-tolerant real-time tasks.

The demand bound function [11, 26] which represents the maximum possible resource demand at any time interval t is defined by:

$$\mathrm{dbf}(W, EDF, t) = \sum_{T_i \in W} \left\lfloor \frac{t}{p_i} \right\rfloor . e_i \tag{2.2}$$

Where p_i is the periodic time period for all $\tau_i \in W$, and e_i is the execution time.

The x-axis in the supply and demand bound functions represents "Time Interval" (as shown in Figure 2.1), which is the time difference between two time points. The demand bound function is a discrete function (repeats after the hyper period) representing the maximum possible demand required during a specific time interval. The demand bound function and supply bound function is based on the time interval, and they are useful because they represent the worst-case resource supply and demand. The supply bound function is a continuous function because supply is always coming, representing the minimum resource supply needed by the system.

Many systems, particularly control systems, are tolerant of individual deadline misses, and considering them as hard real-time would lead to unwarranted pessimism and even resource over-provisioning [23]. Therefore, the delay-tolerant real-time model attempts to capture the system requirements by examining the constraints under supply and demand bound functions that ensure that only bounded delay can be tolerated, as long as the actual delay is less than the known worst-case tolerable delay of the control systems. We provide a novel schedulability analysis test for constrained-deadline periodic real-time systems with the earliest deadline first scheduling algorithm.

2.3 Using power requirements in calculating demand

Embedded systems have different components connected, which do not necessarily require consuming the same amount of power. Therefore, the demand for these components can vary. Guy et al. [27] demonstrated how processor speed level requirements could be added to the demand bound function to select an efficient Dynamic Speed Scaling (*DSS*) level to reduce overall power consumption.
Power-aware demand bound function (pdbf) is provided by $\text{pdbf}_{\text{W}}{}^{1}\text{A}, \text{t}, \text{s}^{\text{o}}$ under scheduling algorithm A = EDF, where the processor speed level s scaled down during the execution of the task and is presented by:

$$\text{pdbf}_{\text{W}}{}^{1}\text{EDF}, \text{t}, \text{s}^{\text{o}} = \tilde{\text{O}}_{\tau_i 2 W} \text{b}\frac{t}{p_i}\text{c}.\frac{e_i}{s} \tag{2.3}$$

Currently, pdbf only considers processor speed level s. Therefore, selecting an efficient processor speed level can be further improved by adding the delay-tolerant level of various components found in the CPS system. For example, the real-time control model can benefit from our work concerning power efficiency because of efficient resource provisioning.

2.4 Resource provisioning for delay-tolerant RTS

Many systems, including control systems, are treated as hard real-time. However, they tolerate individual deadline misses, resulting in unnecessary pessimism and possibly over-provisioning resources, thus consuming more energy than needed. Over-provisioning means the resources are always available by having resource supply always greater than workload demand. This thesis explores avoidance or minimization of resources over-provisioning that is not needed for systems that tolerate short bounded delay.

Delays in supply and demand bound functions is said to occur at any time interval t when dbf_{1t^o} sbf_{1t^o} as shown in Figure 2.2. Delay can be defined by specifying the points of interest (i.e. the t_o overload point and the t_r recovery point) are points of intersection that occurred before dbf exceeds sbf [9], as shown in Figure 2.3.

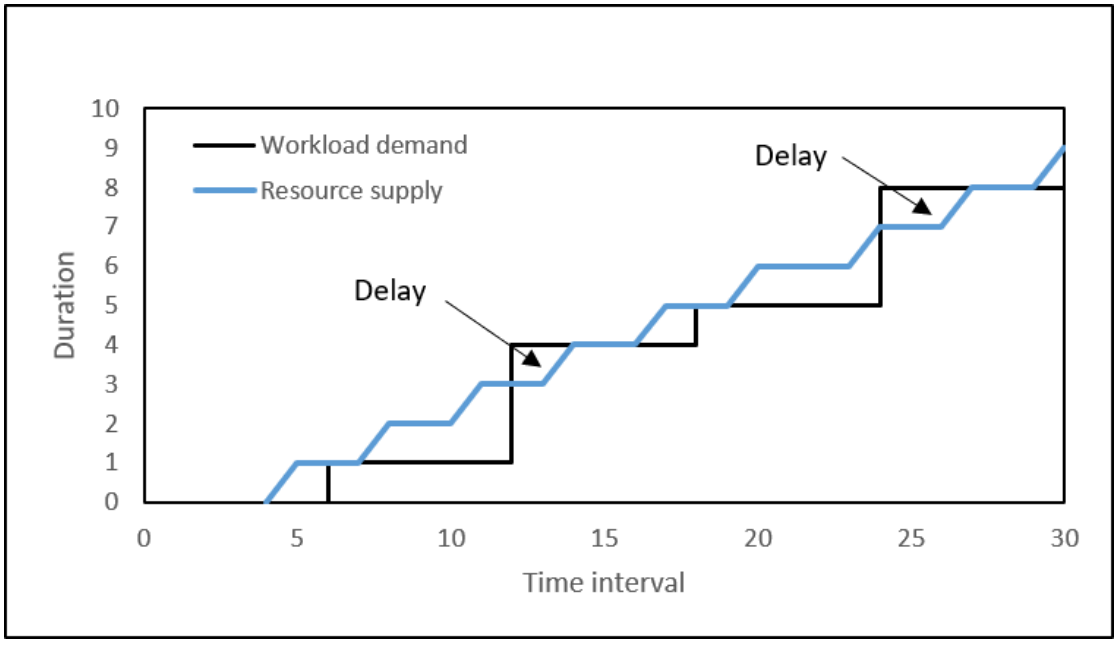


FIGURE 2.2: Example of delays in supply and demand bound functions.

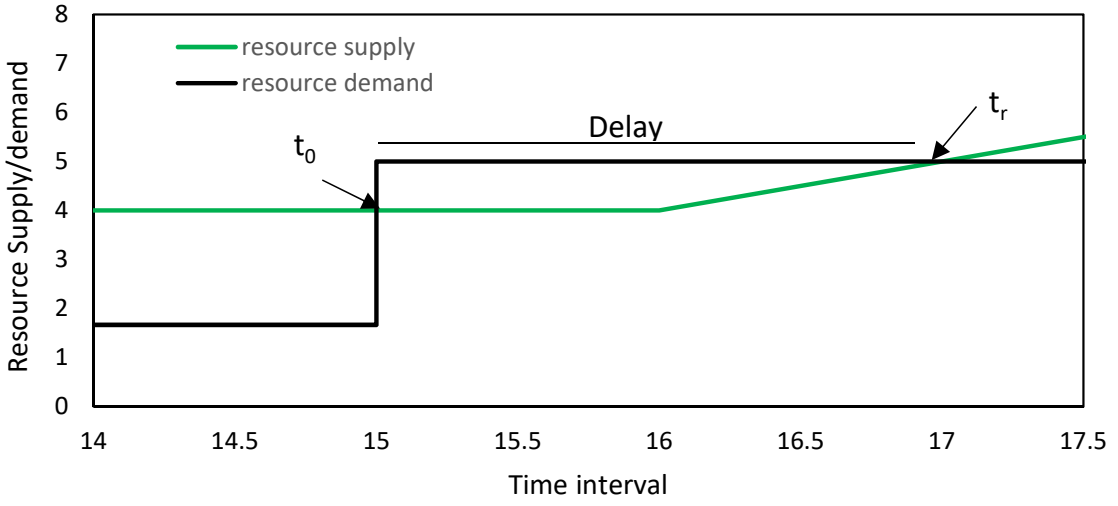


FIGURE 2.3: A closer view of delay.

The delay can be calculated by:

$$D = t_r \quad t_o \tag{2.4}$$

2.5 Related work

2.5.1 DVFS Model

On a Dynamic Voltage Frequency Scaling processor, the power consumption function $P^{1}s^{o}$ of processor speeds may be split into two parts: dependent power P_{dep} that depends on the processor speed and is affected by short circuit and dynamic power consumption that results from charging and discharging CMOS circuit [28]. The dynamic power consumption P_{dyn} that results from gate switching can be represented as a convex function of the processor speed s.

$$P_{dyn} = C_e \quad V_{dd}^2 \quad s \tag{2.5}$$

Where C_e denotes the effective switch capacitance and V_{dd} is the supply voltage. The power consumption of a short-circuit is proportional to the supply voltage. As a result, the speed-dependent power consumption function P_{dep} is convex and increases as the selected speed increases.

There is considerable work that has been studied based on the DVFS mechanism. The real-time dynamic voltage scaling (RT-DVS) problem has been well-studied, and a significant variety of static and dynamic DVS systems have previously been developed.

Aydin et al. [14] investigated the viability of DVS to minimize CPU power usage in Hard RTS. There are three main components of the tested DVS: a static offline mechanism that considers the worst-case workload, a Dynamic Reduction Algorithm (DRA) that lowers power consumption by changing the speed of actual workload speed, and an active mechanism for speed reduction that reduces speed by possible early completion of future executions.

An energy-efficient scheduling algorithm for mixed workload real-time systems slack stealing DVS(SS-DVS) was suggested by Chen et al. [29]. The SS-DVS does not need to know the workload or worst-case execution time of aperiodic tasks in advance. It considers the response time of periodic tasks, making energy usage and response time critical key metrics. The proposed OPT-TSDS algorithm

minimizes the processor speed to a level that makes no gap between supply and demand bound function.

Pillai and shin.[15] proposed two algorithms of DVS. (1) Cycle-conserving RT-DVS: in this algorithm, they assume worst-case execution time initially by making the conservative assumption that the task needs to be released for its next invocation. After the task is completed, they compare the actual processor's cycle to the worst-case specifications. Any unfinished processes that have been split into tasks will be lost and thus, idling the processor. so they take advantage of the DVS algorithm by reducing the frequency of the operating processor. (2) Look-Ahead RT-DVS: attempts to postpone completing the task and set the operating processor's frequency to meet the minimum requirements for the job needed to satisfy all potential deadlines. Therefore, using this approach will allow the operating frequency to run at its maximum to complete all the tasks that have been delayed. However, if the task's execution time takes less time than the worst-case execution time, the device may run at low frequency and voltage such that the maximum timing requirement for the delayed work would never be needed.

Guy et al.[27] present the supply and demand bound functions to reduce energy consumption by proposing an optimal static DVS for task, component, and system levels. Moreover, dynamic DVS for component and task level for better energy savings by taking advantage of unused slack time and resource availability.

Zhang and Guo [21] combined the two existing power management techniques, DVS and Dynamic Power Management (DPM). They proposed an energy-efficient approach using a dynamic low-power sporadic scheduling algorithm where energy consumption is minimized by evaluating the speed of the tasks while still guaranteeing their deadlines.

Rehaiem et al. [19] propose a combination of the DVS and Neural Feedback Scheduling (NFS) with the Priority Energy Earliest Deadline first (PEDF)

algorithm. The neural network structure (i.e., network mode, number of input and output nodes, the number of hidden layer nodes, the transfer function, etc.) was developed based on other researchers' experiments. The start time, the voltage needed to complete each task, and the execution speeds were determined, and a schedule to complete all tasks before the deadline with the minimum amount of power was developed. A list of all tasks "ready list" was created. First, the task with the nearest deadline was determined and was executed at a lower voltage, provided that the deadline could be met. After that, the PEDF chooses the new task, which has the nearest deadline to be executed. Results showed that the proposed technique could reduce the power consumption of embedded systems.

Seawong et al. [20] show four different DVFS techniques for power savings in embedded systems. The four techniques are (1) System Clock Frequency Assignment, where a fixed frequency is used for each task set, (2) Priority Monotonic Clock Frequency Assignment, where each fixed frequency is used for each task, (3) Optimal Clock Frequency Assignment, which saves power through finding the optimal clock frequency using non-linear optimization techniques, and (4) Dynamic (PM) Clock, which assigns frequencies according on the actual execution time needed to complete the task.

A power-aware scheduling algorithm for a bag of task applications proposed by Kim et al.[30], where all the sub-tasks should finish their executions before the deadline. The proposed algorithm selects an appropriate supply voltage for the tasks' processing to minimize the energy consumption based on a cluster system. The cluster system consists of several processing elements (PE) and a resource controller. Users submit their jobs, and the resource controller allocates the jobs based on the processing performance of each PE in terms of (MIPS). The resource controller accepts or rejects to assign the jobs based on their deadlines.

We have summarized some of the related work on energy-efficient techniques based on dynamic voltage frequency scaling as shown in table 2.1, including the algorithms and various characteristics. We do not perform the complexities

presented in the table; it is done by the authors mentioned in the table. Moreover, we added our proposal into the table to understand what is missing in the literature.

TABLE 2.1: Summary of different DVS techniques for Energy savings.

Reference	Periodicity	Scheduler	Complexity	Delay-tolerant	Key contribution
[30]	periodic	EDF	*O(1)*	no	Proposed Pshare-DVS technique that selects fixed static supply voltage throughout the simulation.
[29]	periodic/aperiodic	EDF	*O(1)*	no	Demonstrated SS-DVS method that uses multiple voltages, and the transitions between them cause a neglected overhead.
[15]	Periodic	EDF	*O(n)*	no	The proposed cc-EDF allows the operating frequency to run at its maximum, to force the task finishes its execution time before the deadline.
[27]	Periodic	EDF/RM	*O(m log n)*	no	The designed OPT-TSDS minimizes the processor speed to a level that makes no gap between supply and demand bound functions.
[2],[1]	Periodic	EDF	$O^1 n^{o2}$	yes	The proposed Dynamic-DSS technique minimizes the processor speed to a level that reduces over-provisioning the resources.

2.5.2 Offloading decision techniques

Energy reduction techniques that are based on DVFS have seen significant energy savings but are still constrained by limited resources such as memory capacity, network bandwidth, processor speed, and battery power. These constraints prevent local devices from widely running very complex mobile applications with heavy multimedia and signal processing [31]. Moreover, the mobility needs of such devices impose restrictions on processing capabilities and battery-based power supply, as compared to server computers [32]. In order to achieve energy-efficient task execution in local devices, computation offloading is becoming a more interesting method for moving task execution from restricted-local devices to more powerful remote cloud resources or nearby cloudlets. Unfortunately, offloading mechanism can't be applied for hard real-time tasks because they have hard deadlines, and

such a delay can lead to catastrophic failure. Various aspects such as battery life, current memory capacity, latency, calculation execution time and runtime migration cost should be evaluated before offloading the computation. If all of these factors show that computation offloading reduces power consumption, only offloading is permitted; otherwise, the computation can be done at the local device. Many approaches and frameworks have been presented over the years to make computation offloading decisions feasible. These approaches are presented to improve local devices' performance and make computation offloading more practical by offering additional storage space, bandwidth, and processing power, therefore extending battery life by lowering energy consumption.

Energy reduction has also become a hot issue even in cloud computing, where considerable computing power is required. Energy-efficient offloading policies aim to minimize the energy consumption of real-time systems, which is done by reducing the computational overhead of tasks through computation offloading. As a result, power-intensive tasks are performed in the cloud.

lin et al. [33] proposed a task scheduling algorithm to minimize the total energy consumption of an application. But it did not consider CPU utilization in the task offloading decision.

Xiang et al. [34] proposed online algorithms for near-optimal decision making for offloading requests for multiple tasks, thus reducing energy consumption and transmission time while maintaining performance.

In [32], the authors present an energy-efficient offloading decision process and an offloading dispatcher that efficiently manages all devices' computation and communication resources to dispatch applications to the appropriate device.

Various computation offloading systems are primarily defined based on decisions made statically or dynamically. The static offloading decision, also known as offline profiling, occurs when the task is partitioned during its execution. The static offloading uses performance prediction models to estimate the system's performance and is only valid if it knows which task should be offloaded. The main advantage is to reduce the latency. Wang et al. [35] proposed a static

approach by partition the task into sub-tasks where some tasks are offloaded to the cloud, and some are executed locally based on the information about execution time and data transmission. In [36], the authors propose a complete computation offloading based system where a crucial parameter is considered, which is the user's delay tolerance threshold. The average task execution time running on the local device is calculated and compared to the user's delay tolerance threshold. If the delay tolerance is larger than the execution time, they offload the task; otherwise, the computation is done at the local device.

While dynamic offloading decision occurs when the program is partitioned at run time and also known as online profiling and use optimization decision-making techniques. In [37] transitioning to runtime, the model is presented, an energy-optimal application partitioning technique is proposed, which integrates the resource model by taking into consideration the cost caused by execution time and energy parameters. During compilation, the software partitioning is optimized by turning it into an integer linear problem. A method is proposed that provides a dynamic offloading mechanism that considers application execution behavior. In [38], the whole history of the execution pattern is profiled, and this information is later used to make the best computation offloading option possible. The choice to offload computations is made for each static resource (offloads most used tasks) and dynamic (offloads only invoked jobs). This technique is used when there are many running tasks and makes the application executes faster.

Chapter 3

Proposed approach

3.1 System model and assumptions

We present the types of real-time tasks, workload, and schedulability analysis, execution time, and power models in the system model. Moreover, we introduce some notations and assumptions to be used in this paper.

Two real-time systems can be categorized based on their delay tolerance: (1) Hard real-time system (HRS) cannot tolerate any delay ($\sigma = 0$). If the system becomes overloaded in HRS, the schedulability test indicates insufficient resources exist because dbf exceeds sbf at a given time interval t. Therefore, HRS should always be performed on the local embedded devices because of the negligible contact latency [39]. (2) Delay-tolerant real-time systems (DRS) can tolerate a transient bounded delay. The schedulability of the system is guaranteed by having $D \quad \sigma$. For instance, the feedback control model experience some delay in the feedback loop [13]. The control model specifications include the maximum tolerable delay of σ that can be tolerated in the system. Moreover, we extend our proposed algorithm by integrating a fog/cloud framework to expand the resources to more powerful servers for execution (presented in the "Experimental analysis" section).

3.1.1 Workload and schedulability analysis model

In a periodic resource model, there is a series of components C where each C is modelled as C = ¹W, A⁰. $^{0}A^{0}$ is the scheduling algorithm that defined as earliest

deadline first (EDF) [40]. W is the workload that has a set of independent tasks, it could be hard or delay-tolerant real time tasks $W = \{\tau_1, \tau_2, \ldots, \tau_n\}$. A set of tasks is modeled as a set of tuples: $\{(p_1, e_1), (p_2, e_2), \ldots, (p_n, e_n)\}$, where the workload for one task is defined as $W = \{\tau_1(p_1, e_1)\}$ that has a time period p_i (we assume the time period is the deadline of the task) and execution time e_i.
In our workload model, we assume the tasks with hard deadlines are high criticality tasks, and tasks that can tolerate delays are low criticality tasks. We haven't considered having a mix of high-low criticality tasks, which could be studied in future work.

3.1.2 Execution time model

The execution time of the task is the time needed for the task to complete its function. The worst-case execution time is the maximum time the task will take to complete its job without missing any deadlines. Most dynamic voltage frequency scaling (DVFS) algorithm uses an optimistic hypothesis of a fully scalable Worst-Case Execution Time (WCET) [16], which is defined as $C_i(s) = C_i \bullet s$, where s is the processor speed. Nevertheless, DVFS has been considered not only on the processor but also on some other hardware frameworks such as the bus and memory, which consume power to operate [41]. Therefore, WCET is decomposed into two scalable portions (fully scalable, and partially scalable) [42] and defined as: $C_i(s_{cpu}, s_{mem}) = C_{i(CPU)} \bullet s_{cpu}$, $C_{i(mem)} \bullet s_{mem}$. This is an effective method for considering scalable and partially scalable portions of the WCET tasks while determining the DVFS algorithm task model. Moreover, this WCET task model can be integrated with any DVFS scheduling algorithm for energy savings.

3.1.3 Power model

Power dissipation in digital CMOS circuits has been studied accurately in the literature and can be modeled by simple equations [16]. CMOS circuits have dynamic power (c f V^2), short-circuit power (V I_{sc}), and static (leakage) power

dissipation (V I_{leak}) .

$$^{1}c \quad f \quad V^{2o} \; , \; ^{1}V \quad I_{sc}{}^{o} \; , \; ^{1}V \quad I_{leak}{}^{o} \tag{3.1}$$

Where c is the loading capacitance, f is the clock frequency, V is the operating voltage, I_{sc} is the short circuit current, and I_{leak} is the leakage current [43]. The power consumption of a server depends on some components such as the CPU, memory, and disk, where the CPU is the main component that consumes energy. Since $s \; \alpha \; f$, where s is the processor speed, and $f \; \alpha \; v^{\eta}$ where $0 < \eta \quad 1$. Therefore, linear change in supply voltage results in a linear change in processor speed and clock frequency and also results in at least quadratic change in power supply [44]. Therefore, dynamic power is the primary source of the total power consumption due to the quadratic relationship between the supply voltage and dynamic power consumption [45], and accordingly, static power is ignored in this current study.
Assume the processor has a Dynamic Speed Scaling (DSS) mechanism, which allows the processor to adjust it's operating speed s between minimum and maximum speed levels $^{1}s_{min} \quad s \quad s_{max}{}^{o}$.
The processor speed adjustment mainly depends on different processor operating modes. Each device can be in one of the following modes:

- Active mode: A device in this mode is performing its job and executing the tasks. The processor speed reaches the maximum (s_{max}) (i.e., full processor speed that is equal to 1), and thus more power is consumed in this mode.

- Standby mode: In this mode, the device is not required to provide any service. Therefore, we consider the processor speed to be minimized from full speed to half speed, which is scaled to 0.5. A small amount of power is still consumed to make the device ready to become active within a short amount of time [46].

- Sleep mode: In this mode, the processor consumes the least amount of power, and the processor speed level is reduced to the minimum level (s_{min}). Thus, it takes more time to switch to the active mode.

The energy consumption of the processor consists of both static and dynamic energy. This work focus on the dynamic energy consumption that is managed by a dynamic speed scaling scheme (DSS) [47]. The dynamic energy is proportional to the processor speed s. Thus, proportional to different processor modes. Therefore, for a component C^1W, A^o where the workload has multiple tasks is defined as $^1\tau_i{}^1p_i, e_i, s_i{}^o$, the total energy consumption of the component is defined by [27]:

$$E_t = \tilde{O}_{T_i 2 W} \frac{H}{p_i}.e_i.{}^1s_i{}^{o2} \tag{3.2}$$

Where H is the least common multiplication (LCM) of all p_i, and $^1s_i{}^o$ is the processor speed level of different processor operating mode.

3.1.4 Cloud communication time and transferring cost analysis

Cloud consumers should consider the trade-offs amongst computation, integration, and communication. While moving to the cloud can significantly reduce the power consumption of the local device, it sometimes increases the cost of transferring data to and from the cloud. Therefore, as we show in the experimental analysis, cloud computing is not always the best choice for saving power. Some applications may not be compatible with cloud computing and insufficient to reduce power consumption, such as applications that are not heavy in their computations or have a small amount of data. Therefore, it is essential to consider the communication time and transfer cost in our model to determine the feasibility of processing the application on the cloud. For simplicity, we consider communication as a task in our system. In our model the workload has a set of tasks $W = f\tau_1, \tau_2, \ldots, \tau_n g$ and each task is given by a time period P_1 and execution time e_1, the workload for a specific task is $W = f\tau_1{}^1p_1, e_1{}^o g$.

The communication overhead is considered as a separate task that also has a period and execution time; the period is defined as how often the task is required to be sent to the cloud which we considered to be once in the whole hyper period (the hyper period is defined as the least common multiple (LCM) of

all the periods of the tasks). For example, we have two tasks in the workload $W = \{\tau_1(p_1, e_1), \tau_2(p_2, e_2)\}$, then we consider the communication task as $\tau_3(p_3, e_3)$ where $P_3 = LCM(P_1, P_2)$, and the execution time is the communication time.

3.2 Energy-efficient techniques for CPS

3.2.1 Energy-efficient periodic resource model for delay-tolerant RTS

This section illustrates the proposed scheme by a workflow, as shown in Figure 3.1. We classify the workflow into two parts. First, the optimal processor speed level, which can tolerate some delay, will be determined. We start with full processor speed, i.e., processor speed level equals 1. After that, the supply and demand bound functions will be checked. If there is a gap between them (i.e., the resource over-provisioning exists) then, we will minimize the processor speed to a value equal to $CurrentSpeed \quad \alpha$, where α is a variable, in our work, we use $\alpha = 0.1$. We check the supply and demand bound functions again. If the gap still exists, we keep reducing the processor speed until we reach a value that permits the demand to exceed the supply for bounded time intervals while satisfying the delay specifications of a given control system. Algorithm 1 shows how to get a processor speed level that offers an acceptable delay. Second, we calculate the delay in our scheme and compare it to the control system's worst-case delay specifications. An optimal processor speed level is reached when we get the condition where $D \quad \sigma$ as illustrated in Algorithm 2. We repeat the algorithm from beginning (assign processor speed level S) only if new tasks arrive at runtime. However, in this current work, we assume that tasks/workloads are fixed and pre-defined.

Our proposed algorithm's calculations are mainly based on the scheduling component's resource supply and workload demand W. The complexity analysis in our algorithm classifies into (a) offline computation complexity, where the delay is calculated using Equation 2.4, by determining (t_r, t_o), and (b) runtime complexity analysis for a large number of tasks. The delay D is the maximum delay the system experience before receiving sufficient resources at offline computation.

D cannot exceed σ to guarantee the tasks' deadlines. The computational complexity of runtime computation for a large number of tasks is $O(N^2)$, Where N reflects the number of all tasks within the periods' least common multiple (LCM). The longest time a task set can be delayed is bounded by the delay bound of the control system σ . Therefore, our proposed algorithm is not complex for a large number of tasks.

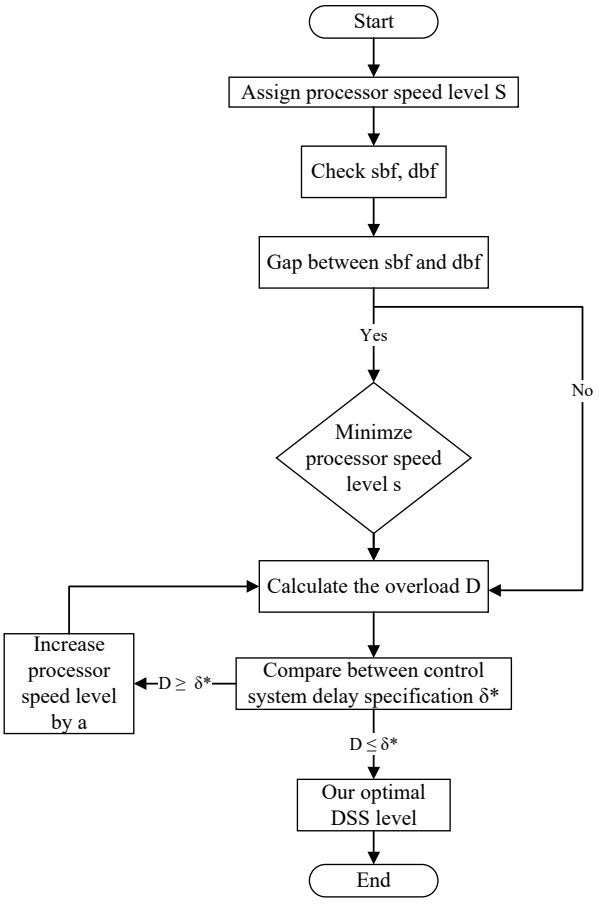


FIGURE 3.1: Work ow of the proposed scheme.

We briefly validate our proposed framework in the following Lemma.

Lemma 3.1. *A scheduling unit that has a component $C(W, A)$ with workload $W = \{\tau_i(p_i, e_i, s)\}$ and $A = EDF$ is schedulable if, and only if, the total workload demand is less than or equal to the resource supply during t provided by periodic resource model R for hard real-time systems [11].*

Algorithm 1 Calculation of an appropriate *DSS* Level.

procedure CALCULATE–DSSLEVEL(S)
  CurrentSpeed S
  t f0, 1, . . . , LCMg
  for each time t_i 2 t **do**
    if t_i 2 »¹k ¸ 1ºΠ 2Θ, ¹k ¸ 1ºΠ Θ¼ **then**
      S»t_i¼ sbf_R¹t_iº = t_i ¹k ¸ 1º¹Π Θº
    else
      S»t_i¼ sbf_R¹t_iº = ¹k 1ºΘ
    end if
    D»t_i¼ $pdbf_w$¹t_iº = $\sum_{T_i 2 b} \frac{t_i}{p_i} c . b \frac{e_i}{s} c$
  end for
  for each time t_i 2 t **do**
    if S»t_i¼ D»t_i¼ **then**
      CurrentSpeed CurrentSpeed α
    end if
  end for
  Delay CalculateDelay¹º
  return Delay
end procedure

Algorithm 2 Calculation of acceptable delay interval.

procedure CALCULATE–DELAY(D)
  DelayOfDSSAlgorithm D
  t f0, 1, . . . , LCMg
  for each time t_i 2 t **do**
    if S»t_i¼ \ D»t_i¼ and S»t_i¼ D»t_i¼ **then**
      D t_r t_o
    end if
  end for
  for each time t_i 2 t **do**
    if D σ **then**
      Optimalspeed CurrentSpeed
    end if
  end for
  Delay CalculateDelay¹º
  return AcceptableDelay
  return Optimalspeed
end procedure

$$pdbf_W(EDF, t, s) = \sum_{T_i \in W} \left\lfloor \frac{t}{p_i} \right\rfloor . \frac{e_i}{s_i} \quad sbf_R(t) \tag{3.3}$$

for all t such that $0 < t \quad LCM$*, where LCM is the least common multiple of the task time period* P_i *for all* $T_i \in W$

To show the sufficiency, we prove the contrapositive, i.e., if all workloads of W are not schedulable by EDF, then Equation 3.3 is false. Let us consider t_2 be the first instance where a job of some workload member T_i of W misses its deadline. Let t_1 be the latest instance where the resource supplied to W was executing a job in which it's deadline comes after t_2. Now the definition of t_1 is a job whose deadline comes before t_2 and which was released at t_1. Since T_i misses its deadline at t_2, the total demand in the workload W in the time interval $[t_1, t_2)$ is greater than the total supply provided by the periodic resource model R in the same time interval $t_2 \quad t_1$.

Lemma 3.2. *The total power consumption for a component* $C(W, A)$ *with* $W = \{\tau_i(p_i, e_i, s)\}$ *and* $A = EDF$ *can be reduced, if the total workload demand exceeds the resource supply during t provided by periodic resource model R for delay-tolerant real-time systems.*

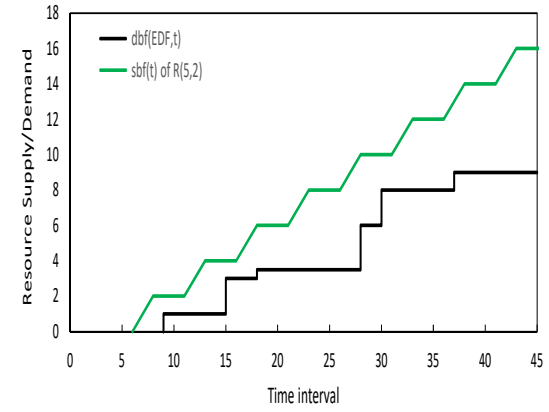
$$pdbf_W(EDF, t, s) = \sum_{T_i \in W} \left\lfloor \frac{t}{p_i} \right\rfloor . \frac{e_i}{s_i} \quad sbf_R(t) \tag{3.4}$$

for all t such that $0 < t \quad LCM$*, where LCM is the least common multiple of the task time period* P_i *for all* $\tau_i \in W$

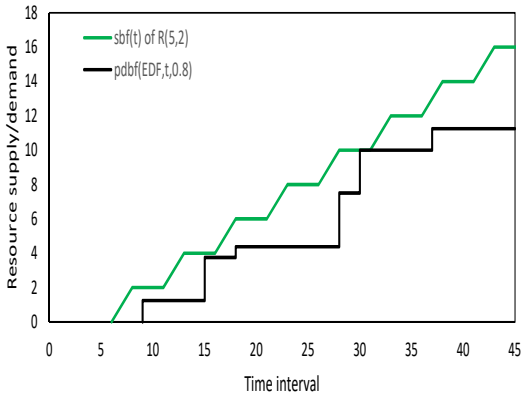
To show the sufficiency, Let us consider a task $(\tau_i(p_i, e_i, s_i))$ running inside a component $C(W, A)$ at processor operating mode speed s_i. The workload demand of a task increases by decreasing the processor operating mode speed. Thus, the task execution time is increased by $e_i \bullet s_i$ at every p_i since it's worst-case execution time e_i is defined at full processor speed level. Therefore, the total energy consumption $E_t = \sum_{T_i \in W} \frac{p}{p_i} . e_i . (s_i)^2$ is reduced by decreasing the processor operating mode speed s_i.

Lemma 3.1 is constructed from [11] where they test the system's schedulability by always having the supply bound function exceeds the demand bound function. Our lemma 3.2 deviates from [11]'s lemma by considering delay-tolerant real-time tasks by accepting the demand bound function exceeds the supply bound function for a bounded time interval.

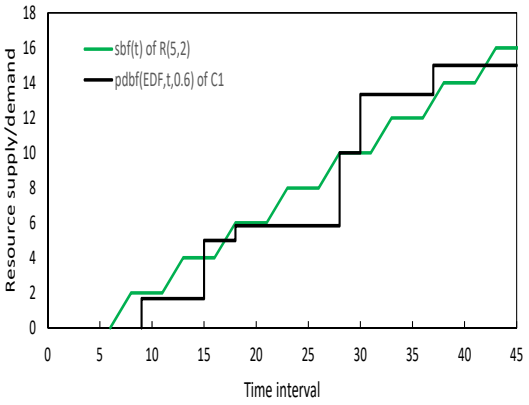
Let us present the following example to illustrate the results of lemma 3.1.



(a) sbf and dbf of C_1 with full processor speed



(b) sbf and pdbf of C_1 with processor speed equal to 0.8



(c) sbf and pdbf of C_1 with processor speed equal to 0.6

FIGURE 3.2: Supply and demand bound functions on di erent processor speed levels

Example 1: Consider a periodic resource supply R(5,2), and a component $C_1 = (W, A)$, where $W = \{(\tau_1 = (9, 1), (\tau_2 = (15, 2)\}$ and A = EDF. In the first case, assuming the processor in the active mode (i.e., full processor speed level), the

minimum resource supply and the resource demand bound functions are analyzed using equations 2.3 and 2.1 and are shown in Figure 3.2(a). It is clear that pdbf$_W$(EDF, t) $\leq$ sbf$_R$(t) and there is a gap between them. This implies the schedulability of the two tasks in W is always guaranteed. The second case, assuming the processor speed level is adjusted to ($S = 0.8$) as shown in Figure 3.2(b), also guarantees the schedulability of the two tasks in the workload as the demand bound function is approximately equal to the supply bound function. In the third case, adjusting the processor speed level to 0.6 results in getting more workload demand than resource supply (see Figure 3.2(c)). The total energy consumed for the first, second, and third case are 11, 7.04, and 3.96 respectively (calculated by using Equation 3.2), that is a total of approximately 64% energy savings.
This total reduction motivates us to study the feasibility of accepting a certain bounded delay in the system.

3.2.2 Computation offloading decision for delay-tolerant real-time systems

Nowadays, mobile computing devices are becoming increasingly popular. They've developed into systems that can perform a variety of tasks. Many of their applications, for example, are computationally intensive, such as video processing, image and voice recognition, etc. [48]. Although mobile device performance will continue to increase, their computation capabilities will remain constrained due to resource limits on these devices that have restricted assets such as processor performance, battery life, and storage capacity [49]. These limitations might be addressed by moving the overwhelming computations from the resource-constrained device to a more powerful server to perform these power-consuming tasks [50].

We now discuss an overview of how the offloading decision is taken, considering a system consists of two-tiered architecture: Local embedded processor and Fog (cloudlet)/cloud. Having a set of tasks categorized based on their delay tolerance:
1- Hard real-time systems (HRS) cannot tolerate any delay and must finish its execution by the assigned deadline. An example of hard real-time tasks could

be an automated braking control system. If these tasks are delayed, a loss of life or catastrophic failure may occur. Therefore, hard real-time tasks always execute on the local embedded processor. Local embedded processors are assumed to be the fastest in their computation due to the negligible communication delay. But have limited computation resources so that they can accommodate a limited number of tasks.

2- Delay-tolerant real-time tasks (DRS) can be defined as weakly hard real-time tasks and soft real-time tasks. This type of task can tolerate a certain amount of bounded delay. However, missing too many deadlines will degrade the system's performance. An example of weakly hard real-time tasks could be motion planning in a dynamic environment. Therefore, only a limited amount of delay can be tolerated. This delay D is an integer number and can be chosen based on the application requirement. Weakly hard real-time tasks are assigned to fog processors because it has a tight bounded delay range. Fog processor (also known as edge node or cloudlet) brings faster response time than cloud computing. It also brings the functionality of cloud computing within the local network but close to the users. However, fog nodes have limited computation resources compared to the cloud. This fog computing model can be used only for weakly hard real-time tasks (WRT).

Soft real-time tasks (SRT) are more delay-tolerant than (WRT). If applications like social media or video playback fail to meet their deadlines successfully, this will not cause a loss of life or system failure; it will only affect the user's experience by having a time lag. These soft real-time tasks are sent to the cloud for execution because cloud servers have a wider delay range than cloudlets. The distance between cloud data processing centers and client devices causes some delay, making cloud computing only beneficial for soft real-time tasks.

Table 4.7 shows the differences of cloudlet and cloud in terms of the computation delay, and the tolerable task size. Cloudlets have computation rates lying between 3000-5000 Million Instructions per Seconds (MIPS). The cloud processor capacity has been taken as 80000 or more MIPS. These values are taken from previous work [51] which represent an overview of how powerful the cloud is compared to

scalable

TABLE 3.1: Values of delay and processing capacity for cloudlets and cloud.

Processor type	Delay	Processing capacity
Fog (cloudlet)	10 ms	3000 - 5000 MIPS
Cloud	100 ms	80000 MIPS

fog nodes. But the communication delay from the user to the fog processor is 10 milliseconds, which is less than the communication delay from user to cloud. Therefore, SRT can best use cloud computing functionalities such as powerful machines, unlimited storage, processing capabilities, and reduced costs.

Algorithm 3. summarize the classifications of real-time tasks.

Algorithm 3 Classifications of real-time tasks

1: HRS = $\{\tau_{1h}, \tau_{2h}, \ldots, \tau_{nh}\}$
2: DRS = $\{\tau_{1d}, \tau_{2d}, \ldots, \tau_{nd}\}$
3: **if** τ_i 2 *HRS* **then**
4: Execute Locally
5: **else if** τ_i 2 *DRS* **then**
6: Execute in the Fog nodes (cloudlet) or cloud
7: **end if**

3.2.2.1 The offline computation offloading decision algorithm

This algorithm decides whether or not to offload the computation of the application from local computing devices onto the cloudlets, based on the supply and demand bound functions for bounded delay-tolerant real-time systems.

Figure 3.3. illustrates how the offloading decision works for a given application, and four steps are needed to make the offloading decision.

Step 1: Run an application with a heavy workload that has more than one task.

Step 2: Analyze the supply and demand bound functions of the application as shown in Figure 3.2(a).

Step 3: If the workload demand exceeds the resource supply, as shown in figure 3.2(c), this means the application drives the system to be overloaded.

Step 4: Offload the application that causes a delay to the cloudlet.

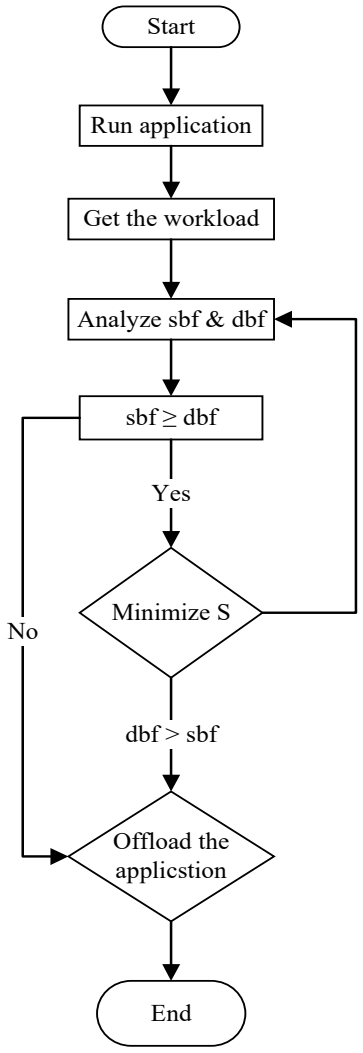


FIGURE 3.3: work ow of o ine computation o oading decision making

As mentioned earlier, ideally, the demand bound function should be less than or equal to the supply bound function. However, a slight change in the system's workload can make the *pdbf* exceed *sbf* at some points in time interval *t* where the delay occurs. More power can be saved if such a delay can be tolerated for bounded delay-tolerant real-time systems where delaying the deadline can be acceptable for a certain amount of time. However, in hard real-time systems, such an overload cannot be tolerated. Therefore, hard real-time systems always execute in the local embedded processors to avoid having any delay.

3.2.2.2 The online computation offloading decision at run-time computation

The online computation offloading decision is for unbounded delay-tolerant real-time systems and achieves significant power savings while guaranteeing their schedulability. But it is not feasible if the number of tasks increases where the processing operations can take a longer time to finish their execution [52]. Therefore, we propose a very lightweight algorithm at runtime computation for systems where the latencies are related to the quality of service. Algorithm 4 explains how run-time computation works. An efficient offloading decision is taken based on power consumption and processor usage of the application.

Algorithm 4 Offloading decision at run-time computation.

```
1: procedure CALCULATE–μ
2:     P_cons     PowerConsumption
3:     U_proc     ProcessorUsage
4:     t     f0, 1, . . . , TestFrameg
5:     for each NewTask do
6:         calculate (P_cons.U_proc°     μ
7:         Fixed Threshold     Thr
8:         if μ     Thr then
9:             Decision     Offload
10:        else
11:            Decision     LocalExecution
12:        end if
13:    end for
14: end procedure
```

Many applications are intensive in their computations to be executed on the local devices. The proposed algorithm at runtime computation saves the power of local computing devices by offloading high intensive applications to the cloud. Mainly the offloading decision depends on power consumption $^{1}P_{cons}{}^{\circ}$ and Processor usage $^{1}U_{proc}{}^{\circ}$, based on those two parameters, the decision whether to offload the task into the cloud or execute it locally is taken. As shown in algorithm 4., we monitor the running tasks for a specific amount of time (TestFrame) and get the power consumption (P_{cons}) and processor usage (U_{proc}) of the application and calculate μ. We estimate a threshold Thr and compare it with μ. If μ exceeds the threshold, that means the application consumes a large amount of power, or the utilization

of the processor is considerable. Therefore, an offloading decision is taken, which moves the computation to a more powerful machine to improve the performance of the task and save energy in the local machines.

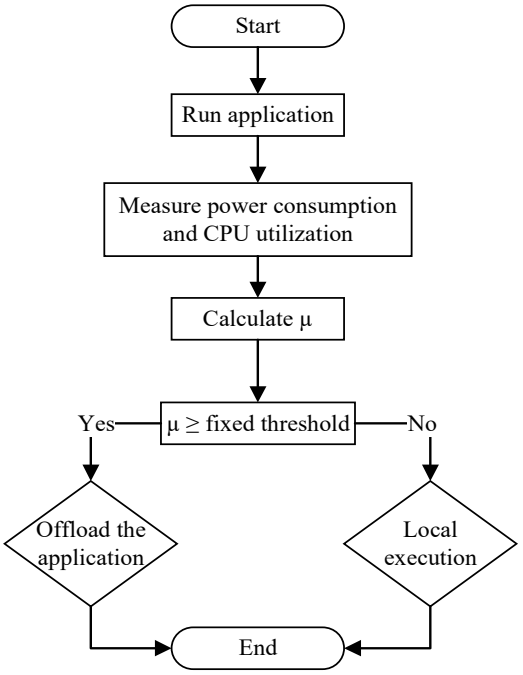


FIGURE 3.4: Work ow of the online computation decision making

To show the feasibility of the proposed algorithm at runtime computations, four steps will be explained on how to make the offloading decision as shown in Figure 3.4, and then an example will be given for better understanding.

Step 1: Monitor the power consumption P_{cons} and processor usage U_{proc} of multiple tasks at run-time to get a value μ equal to the multiplication of the both parameters (P_{cons} and U_{proc}) as shown in Equation 3.5.

Step 2: Determine a threshold Thr value that can be kept for the engineer to fix, in our work, we give a threshold value equal to 5.

Step 3: Compare the threshold determined in the previous step with the μ that was obtained in the first step.

Step 4: The decision to offload the task into the cloud is taken if the μ is greater than or equal to the Thr. Otherwise, keep the task to be executed locally.

$$\mu = P_{cons} U_{proc} \tag{3.5}$$

$$\text{Offload} : \mu \quad \text{Thr} \tag{3.6}$$

$$\text{Locally} : \mu < \text{Thr} \tag{3.7}$$

Example 2: Five running tasks are implemented at the local computing device, and the results of the power consumption, execution time, and processor usage of these five tasks are estimated as shown in Table 3.2. Thereafter, we compare the threshold to the μ value for each task that is executed locally. Finally, a decision to offload the task is taken if the μ is greater than or equal to the given threshold (see Table 3.3), it is clear that for cases 1 and 2, the μ of the local execution is less than the fixed threshold. According to our runtime algorithm, it is better to execute the first two tasks locally rather than send them into the cloud for the power-efficient task. As shown in Table 3.2, it is obvious that the task with the smallest data size and execution time consumes less power. However, it is not necessary to have a processor usage linearly scaled to the power consumption. Therefore, we rely on P_{cons} and U_{proc} as explained in the algorithm 4 and equations 3.5,3.6,and 3.7.

TABLE 3.2: Execution time, power consumption and processor usage of di erent tasks executed locally

	Local Execution		
Data Size (MB)	Power consumption (W)	Execution Time (min)	Processor usage (%)
3	12.86	1	33
6	13.86	1.5	30
10	15	2	36
15	15.87	4	36.5
25	20.8	8	38

TABLE 3.3: Offloading decision based on the calculated μ and fixed threshold of five different cases from Table 3.2.

Case	μ	Threshold	Decision
1	4.24	5	Local execution
2	4.15	5	Local execution
3	5.4	5	Offload
4	5.79	5	Offload
5	7.9	5	Offload

3.2.3 Energy-aware scheduling optimization technique for HRS and DRS

Unfortunately, most real-time scheduling frameworks have focused on efficient scheduling tasks inside the components providing no interest in energy consumption. Minimizing energy consumption for hard and delay-tolerant real-time systems while guaranteeing their timing deadlines is challenging because lowering the processor speed to its minimum achieves the highest energy savings but no longer guarantees the schedulability of the system. Because the minimum the processor speed is, the longer the task execution time, leading to missing timing requirements. Therefore, "critical speed" is crucial, corresponding to a speed that achieves total energy minimization and further lowers the speed beyond that limit, resulting in deadline missing. Therefore, this factor should be considered, and thus, finding an efficient energy-aware optimization technique is essential to achieve energy savings while satisfying the tasks' deadlines. We propose an optimal Dynamic Speed Scaling *DSS* for task and component levels (OPT-TDSS, OPT-CDSS) by adjusting the processor speed to a certain level that efficiently estimates the resource requirements by having the resource supply almost equal to the workload demand. To the best of our knowledge, supply and demand bound functions have not considered in an optimization model,in the literature, to reduce energy consumption for hard and delay-tolerant real-time systems.

Energy-aware scheduling optimization problem is quite challenging because of the inherent nature of DVS scheme itself, that is, processor runs at its full speed

(i.e., $s_i = 1$), which results in over-provisioning the resources by having resource supply always exceeds the workload demand and thus, consumes more energy than required. Moreover, aggressive reduction of processor speed affects the efficiency of the system by missing deadlines. Therefore, our main objective is to formulate the problem as an optimization problem and solve it to achieve the maximum possible energy savings without affecting the schedulability of the workload. The total energy consumption E_t is defined by: $E_t = \sum_{T_i 2 W} \frac{H}{p_i}.e_i.(s_i)^2$, where H is the least common multiple (LCM) for all P_i and s_i is processor speed level for task T_i.

In this work, we propose the optimal task level DSS (OPT-TDSS) model to find s_i for all $T_i \; 2 \; W$.

$$\text{minimize} \quad \sum_{T_i 2 W} \frac{H}{p_i}.e_i.(s_i)^2$$

$$\text{subject to}$$

$$\sum_{T_i 2 W} \left\lfloor \frac{t}{p_i} \right\rfloor . \left\lfloor \frac{e_i}{s_i} \right\rfloor \quad sbf_R,$$

$$0 \quad s_i \quad 1, i = 1, 2, ..., n$$

The optimization problem formulated as three dimensional where the total energy consumption E_t needs to be minimized with a variable s_i and the static values p_i and e_i. The objective function is subject to two constraints: (1) The energy-aware demand bound function cannot surpass the supply bound function to meet the hard deadlines, (2) The Dynamic Speed Scaling DSS allows the processor to adjust its operating speed S between a minimum $s_{min} = 0$ and a maximum speed level $s_{max} = 1$, that is constrained by $0 \quad s_i \quad 1$.

We solve the optimization problem using AMPL/KNITRO solver that uses the Interior-Point/Barrier Direct algorithm, which solves linear and nonlinear convex optimization problems. Interior-point methods (barrier methods) replace the nonlinear programming problem with a set of barrier subproblems controlled by a barrier parameter. Interior-point methods perform one or more minimization

steps on each barrier subproblem, then reduce the barrier parameter and reform the process until the original problem has been solved to the desired accuracy. [53]. Inequality constraints are avoided by adding a barrier term to the objective function, which causes the optimal unconstrained value to be in the feasible space. We also propose a variant of OPT-TDSS, which we call optimized component-level DSS (OPT-CDSS). In OPT-CDSS, the speed level s_i is the same for all tasks. In this case, we replace the variable s_i with s, which is constrained by 0 s 1.

For delay-tolerant real-time systems (DRS), the optimization model still the same but the deadline will be delayed by the worst-case tolerable delay value σ , that is, $p_i = p_i$, σ , the deadline is the time period in our task model p_i. For instance, having a tolerable-delay task $\tau_1{}^1p_1, e_1{}^\circ$, the new task model for the DRS proposed optimization model will be $\tau_1{}^1p_1$, σ , $e_1{}^\circ$. For example, having a workload specification for delay-tolerant task $\tau_1{}^12, 1^\circ$ and worst case tolerable delay $\sigma = 3$, the new workload model will be $\tau_1{}^15, 1^\circ$.

Chapter 4

Experimental results and analysis

4.1 Goals of the experiments

In our experimental work, we conduct several experiments to illustrate the applicability of the proposed framework regarding the resources reservation in delay-tolerant CPS. The goals of our experiment are to reduce energy consumption while meeting the timing requirements. Therefore, we conduct three experiments to determine how the proposed approaches achieve considerable energy savings without compromising the timing requirements.

First we evaluate the performance of the proposed energy-efficient Dynamic Speed Scaling (DSS) for delay-tolerant CPS under two feedback control models. We use the dynamic *DSS* level over the static resource supply model. After that, we use the optimal *DSS* level to get a dynamic resource supply until we reach the optimal static resource supply with a static *DSS* level that achieves the highest energy savings for the used control models. The goal for this experiment is the following:

> Goal: Given a periodic resource model with a real-time control system workload W = $\{\tau_i(p_i, e_i)\}$ and a maximum tolerable delay σ . Find an appropriate *DSS* level that experience a delay *D* when (*dbf*) exceeds (*sbf*) with at most σ .

Second we evaluate the feasibility of using the proposed offloading decision approach for delay-tolerant CPS. We use the powerstat monitoring tool to quantify

energy consumption and CPU utilization for four soft real-time tasks. We investigated the technical and economic feasibility of using offloading to extend various processes requiring different computational power to the cloudlet or cloud for considerable energy savings. The goal for this experiment is the following:

> Goal: Given a periodic resource model with delay-tolerant workload $W = \{\tau_i(p_i, e_i)\}$ and a worst-case delay σ . Find an appropriate *DSS* level that permits the resource supply to drops below the workload demand and offload the delay-tolerant tasks when the actual delay D exceeds σ .

Third we evaluate the proposed energy-aware optimization model (OPT-TDSS and OPT-CDSS) for hard real-time systems using supply and demand bound functions under a set of avionics benchmarks. We use AMPL/KNITRO optimization solver that minimizes the energy consumption under energy and timing constraints to guarantee that the tasks never miss the deadline. We assume that the workload is scheduled using the earliest deadline first (EDF) scheduling policy. The goal of this experiment is the following:

> Goal: Given a hard real-time system with workload $W = \{T_1, T_2, \ldots, T_n\}$ with a periodic resource supply, find an optimal processor speed s_i for each task T_i to maximize the overall energy efficiency under the EDF scheduling policy.

4.2 Energy-efficient periodic resource model performance for delay-tolerant RTS

In the current experiment, we used the real specifications of two feedback control models, the inverted pendulum and chemical distillation models $W = \{\tau_1(2, 10), (15, 10)\}$ as shown in Table 4.1. The workload is scheduled under the earliest deadline first scheduling algorithm. Those two control models have a maximum tolerable delay $\sigma = 3$ [9], where the system will not be schedulable if the actual delay D exceeds σ .

TABLE 4.1: Resource specifications for feedback control model under worst-case tolerable delay σ = 3.

tasks	Workload		Resource	
	p_i	e_i	$_i$	θ_i
τ_1	2	1	2	1
τ_2	15	1	2	1

This experiment has been performed using Matlab and clarifies more details than presented in example 1, performed by hand 3.2.1. We run our simulator multiple times under four different processor speeds to get the optimal speed level by having D slightly less than or equal to σ . We assume a constant resource supply R¹1, 1º, while dbf¹EDF, tº changes by changing the processor speed level.

First case, we run our model at full processor speed level $s = 1$, we get dbf¹EDF, tº less than sbf_R¹tº without any occurrence of delay as shown in Figure 4.1 and Table 4.2. Therefore, at $s = 1$, the system is schedulable and more power can be saved by minimizing s. The energy consumed in this case is 17 energy units (calculated using the energy equation 3.2).

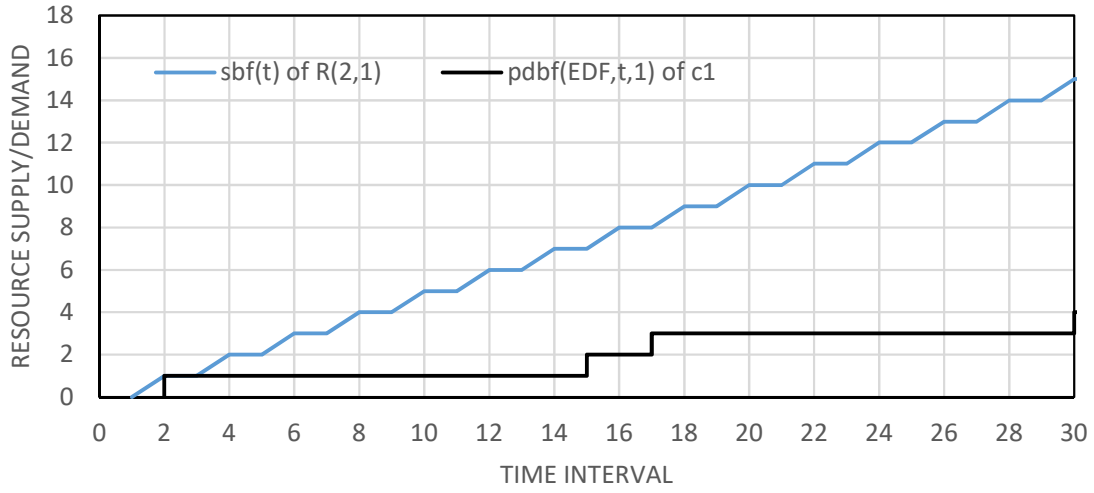


FIGURE 4.1: Supply and demand bound functions (sbf, pdbf) of C_1 with full processor speed $s = 1$

Second case, we minimize the processor speed level from i.e. S = 1 to S = 0.7. As shown in Figure 4.2, we compute *sbf* , *dbf* , and D, the system becomes overloaded by having dbf_A¹w, tº exceeds sbf_R¹tº during certain amount of time (See Table

TABLE 4.2: Resource speci cations for feedback control model at full processor speed level ($s = 1$) under worst-case tolerable delay $\sigma = 3$

Time	*dbf*	*sbf*	delay ($d = t_r \quad t_o$	schedulability test
2	1	1	0	d σ , schedulable
15	1	7	0	
17	2	8	0	
30	3	15	0	

4.3). The delay D = 1.1 (calculated using Equation 2.4.), which is less than σ . The energy consumed at this case is 8.33 energy units (calculated using Equation 3.2), which obviously is decreased by 51% compared to a full processor speed level.

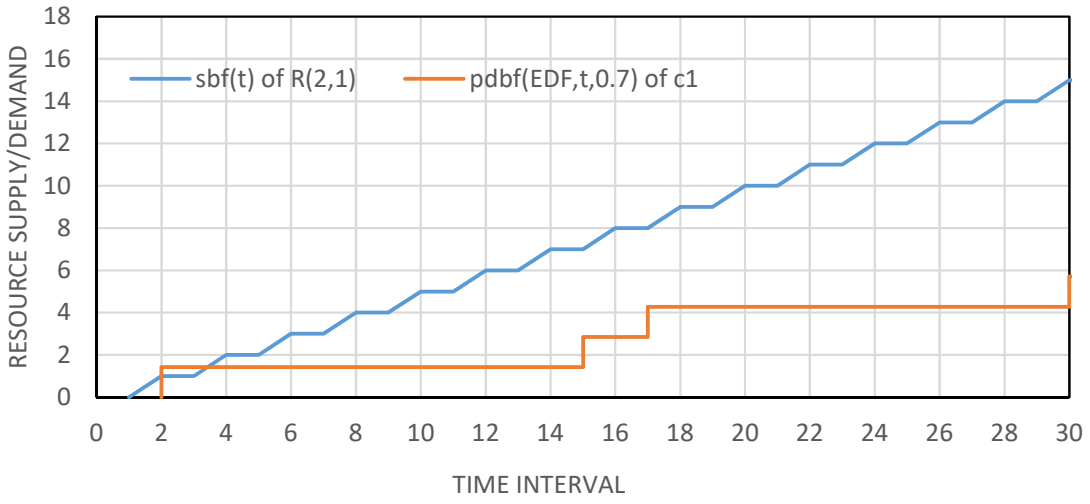


FIGURE 4.2: Supply and demand bound functions (sbf, pdbf) of C_1 with processor speed $s = 0.7$

TABLE 4.3: Resource speci cations for feedback control model at processor speed level ($s = 0.7$) under worst-case tolerable delay $\sigma = 3$

Time	*dbf*	*sbf*	delay ($d = t_r \quad t_o$	schedulability test
2	1.42	1	1.1	d σ , schedulable
15	1.42	7	0	
17	2.85	8	0	
30	4.2	15	0	

Third case, we scale down the processor speed level from $s = 0.7$ to $s = 0.6$, as shown in Figure 4.3. The results of our simulations, show that $dbf_A(w, t)$ exceeds $sbf_R(t)$. At this processor speed level $s = 0.6$, the delay is equal to 1.8, which

means the system is still schedulable because we have D σ (See Table 4.4). Further, the total amount of energy consumed at this DSS level equals 6.12 energy units.

Fourth case, we scale down the processor speed level from $s = 0.6$ to $s = 0.5$, as shown in Figure 4.4. The results of our simulations, show that $\mathrm{dbf}_A(w, t)$ exceeds $\mathrm{sbf}_R(t)$. At this processor speed level $s = 0.5$, the delay is equal to 2, which means the system is still schedulable because we have D σ (See Table 4.4). Further, the total amount of energy consumed at this DSS level is equal to 4.25 energy units. If we scale down the processor speed level to $s = 0.4$, as shown in Figure 4.5, the system will no longer schedulable because $D = 3.4$, which is greater than $\sigma = 3$ (see Table 4.5). Consequently, the optimal processor speed level that can tolerate the bounded delay is equal to 0.5. That represents a total energy savings of 73% compared to running this control model at the full processor speed level.

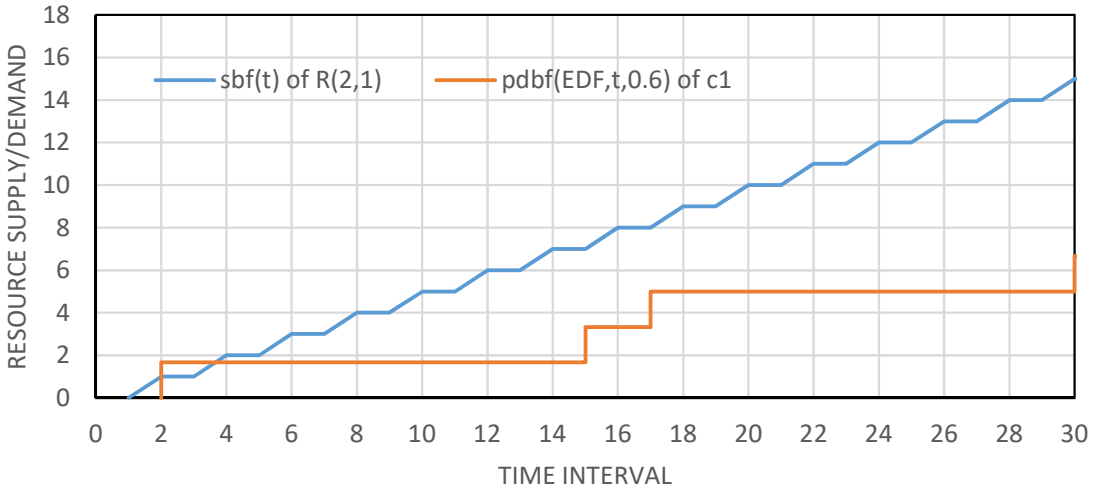


FIGURE 4.3: Supply and demand bound functions (sbf, pdbf) of C_1 with processor speed $s = 0.6$

It's important to compare our work with Tchamgoue's work [27] to show how our proposed algorithm achieves more power savings while satisfying the timing requirements of delay-tolerant CPS. The main difference between our proposed technique and [27]'s method is that we consider delay-tolerant real-time systems. We accept having the supply bound function drops below the demand bound function for a bounded time interval, as shown in Table 4.6, In case 1, Tchamgoue's

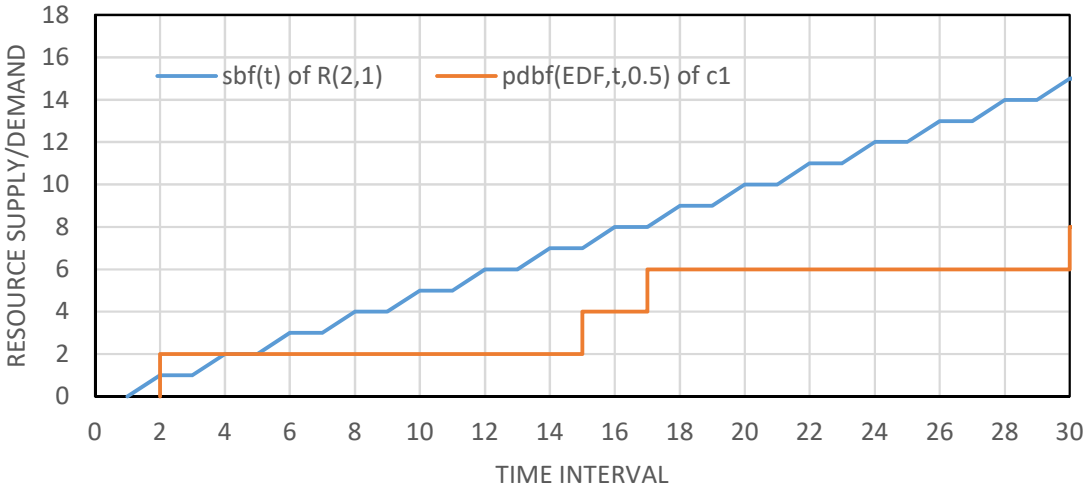


FIGURE 4.4: Supply and demand bound functions (sbf, pdbf) of C_1 with processor speed $s = 0.5$

TABLE 4.4: Resource reservation for feedback control model at processor speed level ($s = 0.6$) under supply and demand bound functions, and worst-case tolerable delay $\sigma = 3$

Time	*dbf*	*sbf*	delay ($d = t_r \quad t_o$°	schedulability test
2	1.67	1	1.8	d σ , schedulable
15	1.67	7	0	
17	3.33	8	0	
30	5	15	0	

TABLE 4.5: Resource reservation for feedback control model at processor speed level ($s = 0.4$) under supply and demand bound functions, and worst-case tolerable delay $\sigma = 3$

Time	*dbf*	*sbf*	delay ($d = t_r \quad t_o$°	schedulability test
2	2.5	1	3.4	d σ , not schedulable
15	2.5	7	0	
17	5	8	0	
30	7.5	15	0	

method [27] and our method are the same, where the tasks' schedulability is guaranteed by having $sbf > dbf$. In case 2, according to [27] by decreasing s from 1 to 0.7, the task is no more schedulable because $dbf > sbf$, and thus, their algorithm can not be continued. However, according to our method, cases 2 and 3 are still valid because they can tolerate a transient bounded delay within an interval no greater than the worst-case delay of the control systems. In case 4, the tolerable

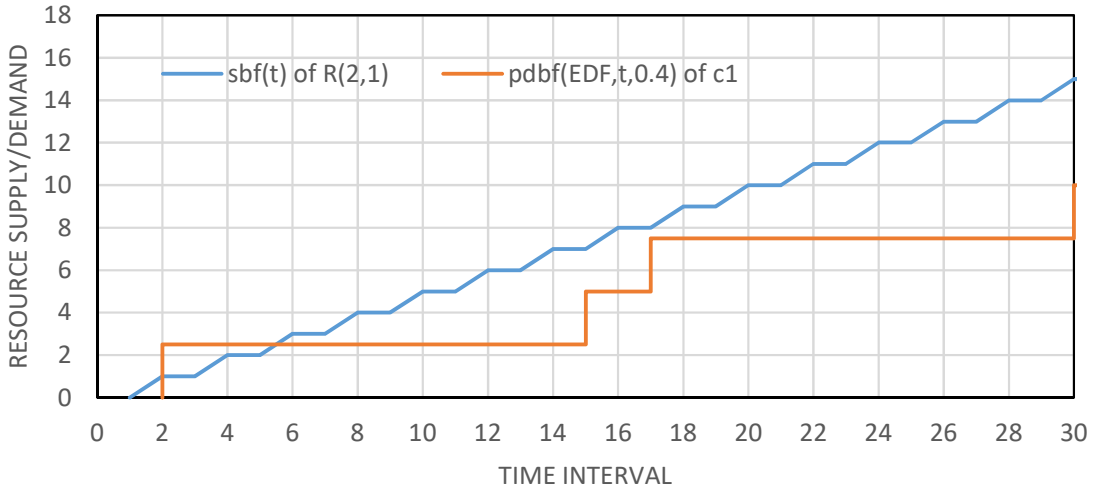


FIGURE 4.5: Supply and demand bound functions (sbf, pdbf) of C_1 with processor speed $s = 0.4$

delay exceeds the worst-case delay of the control system; therefore, the system is not schedulable. Thus, our proposed algorithm is not valid at this level. It's been said that case 4. is optimum in achieving the highest energy savings without missing the deadlines.

TABLE 4.6: Comparative performances between Tchamgoue's work and our proposed algorithm.

Case	Processor speed	Energy consumption	schedulability test	
			Tchamgoue's proposal	Our proposed approach
1	1	17	(*sbf* *dbf*), schedulable	
2	0.7	8.23	(*sbf* *dbf*), not schedulable	(*sbf* *dbf*), schedulable
3	0.6	6.12		
4	0.5	4.5		
5	0.4	2.72		(*sbf* *dbf*), not schedulable

4.2.1 Extending the resources for DRS to the cloud

Offloading the power-intensive computation tasks to cloudlets or cloud servers has been used to improve the performance of delay-tolerant systems. This experiment investigates the feasibility of using the Computation Offloading Decision (COD) approach for systems that can tolerate transient bounded delay to determine when the tasks should be offloaded in order to achieve timely results. At run-time computation, we run four different power-intensive delay-tolerant real-time tasks at the local computational device. We use a laptop with Intel® Core™ i5-6300HQ

CPU @ 2.30GHz, installed memory(RAM) 8GB, 64-bit operating system, and an x64-based processor. We monitor the power consumption and CPU utilization using the Powerstat tool during the computation of those tasks, and we quantify and analyze the results of each operating condition. Amazon Web Service (AWS) offers Amazon Elastic Compute Cloud (Amazon EC2), one of the most secure and cost-effective cloud computing platforms. Amazon EC2 platform is developed and integrated with the computing device (laptop). The same computations/simulations will be offloaded and performed on the Amazon EC2 platform. At offline computation, we get the workload of each application, analyze the resource supply and demand bound functions, and make the offloading decision based on the occurrence of such a delay. Finally, the cost of both cases (cloud and local execution) will be estimated to determine the economic feasibility of using cloud computing for power management in computers and embedded systems.

4.2.1.1 Online COD performance

Powerstat is a tool used to monitor the power consumption and CPU utilization of the computational device that uses a lithium-ion battery as a power source [54]. The output of the Powerstat shows power consumption statistics, and at the end of the running process, Powerstat calculates the average, standard deviation, and min/max of the gathered data. The operating system of the laptop that was used to perform the experiments is Ubuntu Linux 14.04 SP1. The power consumption and CPU utilization of the laptop are monitored and measured under four different computational delay-tolerant tasks (we assume that these tasks can tolerate delays): Idle state Web browsing (WB) Performing a simple calculation using Matlab (MSC) Performing a power-intensive simulation (Simulink modeling) using Matlab (MPI) We first take a baseline of the laptop's power consumption and CPU utilization before the application runs; after that, we compare the results after running the tasks to see the differences. The power consumption and CPU utilization of the laptop under each task are monitored for 480 seconds, the default powerstat monitoring time for a process is 480 seconds. Figure 4.6. shows an example of the Powerstat output, which includes the starting

time for each sample (time), the percentage of CPU that gets to rest (idle), the running processes (run), and the power consumption (watts). Moreover, the average, minimum, and maximum power consumption for the whole 480 seconds.

```
  Time    User  Nice   Sys  Idle    IO  Run Ctxt/s  IRQ/s  Watts

01:46:38   8.9   0.0   0.6  90.5   0.0    1   2337    655  10.97
01:46:48   5.0   0.0   0.5  94.4   0.1    1   1470    462   9.65
01:46:58   8.8   0.0   0.7  90.4   0.1    2   2695    546  10.03
01:47:08   6.0   0.0   0.7  93.3   0.1    1   1978    560   9.99
01:47:18   9.6   0.0   0.9  89.4   0.0    1   3445   1011  10.28
01:47:28   7.1   0.0   0.5  92.4   0.0    1   2377    396  10.38
01:47:38   7.2   0.0   0.3  92.4   0.1    1   1654    417  10.11
01:47:48   2.8   0.0   0.3  96.8   0.0    1   1384    192   9.07
01:47:58   4.6   0.0   0.7  94.7   0.1    2   2036    616   9.31
01:48:08   7.8   0.0   0.8  91.4   0.0    3   3087    907  10.23
01:48:18   7.0   0.0   0.5  92.4   0.1    2   2328    382  10.23
01:48:28   8.7   0.0   0.5  90.8   0.0    1   1722    383  10.24
01:48:38   4.1   0.0   0.6  95.3   0.1    1   2129    615   9.46
01:48:48   8.2   0.0   0.8  91.0   0.1    2   3309    880   9.76
01:48:58   8.4   0.0   0.6  91.0   0.1    1   2573    501  10.43
01:49:08   3.8   0.0   0.3  95.9   0.0    2   1360    213   9.12
01:49:18   2.4   0.0   0.3  97.3   0.0    1   1430    194   8.70
01:49:28   3.0   0.0   0.3  96.7   0.0    2   1356    212   8.60
01:49:38   2.8   0.0   0.3  96.8   0.1    1   1437    290   8.93
^C-------- ----- ----- ----- ----- ----- ---- ------ ------ -----
 Average   6.1   0.0   0.5  93.3   0.0  1.4 2110.9  496.4   9.76
  StdDev   2.4   0.0   0.2   2.5   0.0  0.6  659.6  237.6   0.64
-------- ----- ----- ----- ----- ----- ---- ------ ------ ------
 Minimum   2.4   0.0   0.3  89.4   0.0  1.0 1356.1  191.6   8.60
 Maximum   9.6   0.0   0.9  97.3   0.1  3.0 3444.6 1011.0  10.97
-------- ----- ----- ----- ----- ----- ---- ------ ------ ------
Summary:
  9.76 Watts on Average with Standard Deviation 0.64
```

FIGURE 4.6: Example of the Powerstat output (time, idle, run and watts).

All experiments are conducted three times to check their repeatability. The variations of the power consumption for all experiments were negligible. For instance, the maximum power consumption variation for the idle state operating condition was 5% (i.e., 6.94 W, 6.81 W, and 7.18 W). This confirms that the experiments are repeatable and the results are accurate. Figure 4.7 and 4.8 show the variation of the power consumption and CPU utilization of the laptop, respectively, versus time for the four different tasks.

The minimum, average, and maximum power consumption of the laptop under the four operating conditions are shown in Figure 4.9. As expected, the highest power consumption of the computer occurred when Matlab is used to perform a power-intensive simulation (Simulink modeling) (MPI). On the other hand, the power consumption of the laptop while performing Matlab simple calculations, web browsing, and idle state are negligible.

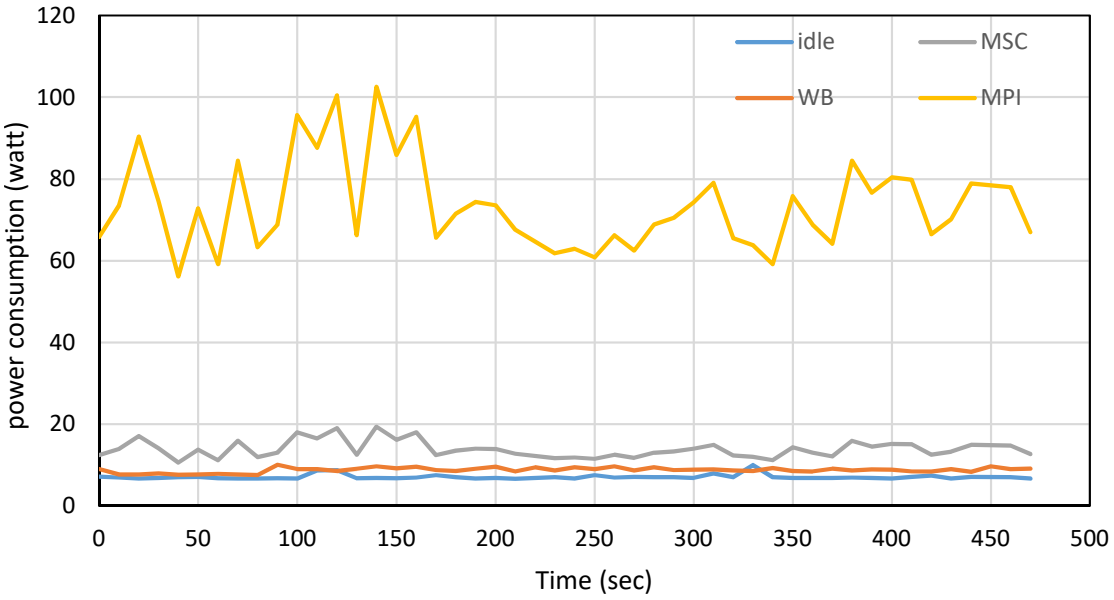


FIGURE 4.7: Power consumption of the laptop versus time for the idle state, WB, MSC , and MPI operating conditions.

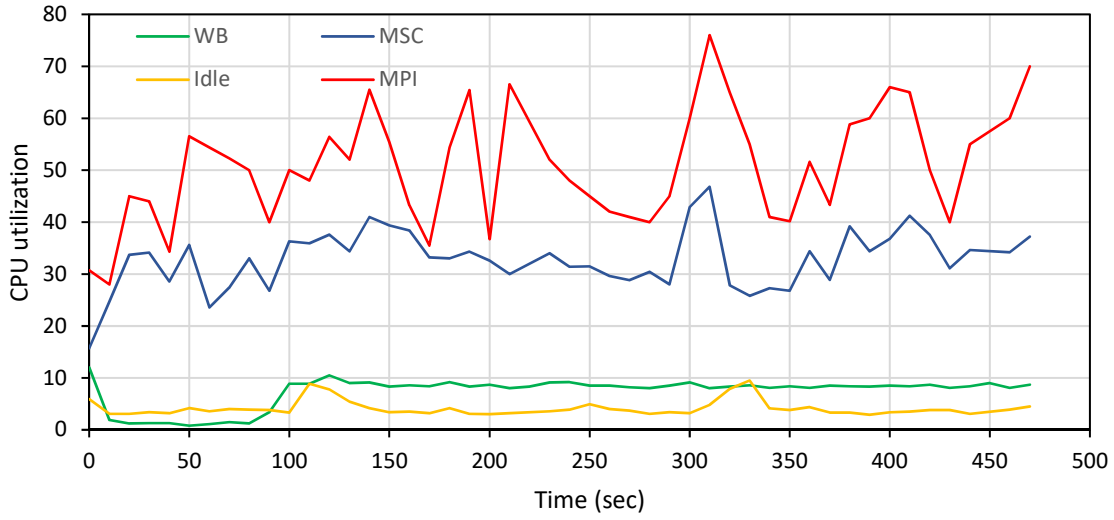


FIGURE 4.8: CPU utilization ¹%º of the laptop versus time for the idle state, WB, MSC, and MPI simulation (Simulink modeling) operating conditions.

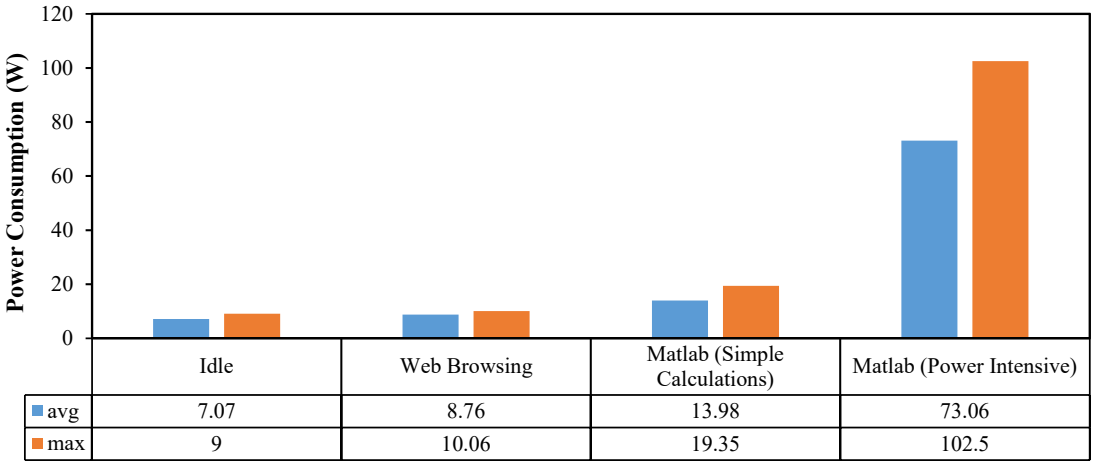
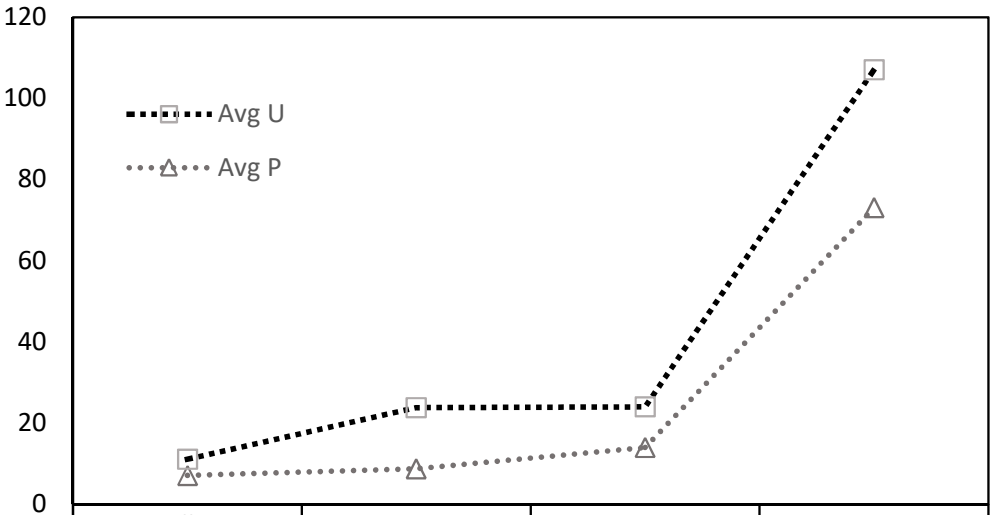


FIGURE 4.9: Comparison between the average and maximum power consumption of the laptop for idle state, WB, MSC, and MPI operating conditions.

In the real world, different workload works not only in varied power consumption during its work time but also in a varied CPU utilization according to the characteristics of the tasks [55]. Therefore, using only the power consumption to decide whether to offload the application into the cloud or not is insufficient. By running the four computations on the local computing device, we notice that not always as the power consumption increases the CPU utilization increases. As shown in Figure 4.10, the processor utilization of MSC decreased; however, the power consumption increased. This implies the importance of combining CPU utilization with power consumption, as discussed in the run-time computation algorithm (section 3.2.2.2). Based on the values we have for the CPU utilization and power consumption, we calculate the μ for each operating condition (idle, WB, MSC, and MPI) using Equation 3.5. Depending on a given threshold, in this experiment, a threshold is fixed, which is equal to 150, the decision is taken to send an offloading request to execute MPI on the cloud. The annual power consumption (8760 hours) of the laptop while performing MSC and MPI are 93 kWh and 835 kWh, respectively (see table 4.10). Therefore, using Matlab to perform power-intensive simulation consumes a large amount of power, and thus it is a useful application for cloud computing. Results show that significant power can be saved by executing power-intensive applications into the cloud. Using cloud computing can also be beneficial for large companies that run complex simulations all

year round. Furthermore, other embedded systems (e.g., smart cell phones, smart thermostats, etc.) can also benefit from cloud computing.



	Idle	WB	MSC	MPI
Avg U	400%	1500%	1000%	3400%
Avg P	7.07	8.76	13.98	73.06

FIGURE 4.10: CPU utilization [%] and power consumption in watt of the laptop versus time for the idle state, WB, MSC, and MPI.

4.2.1.2 Offline COD performance

A simulation is performed for MSC and MPI to show that having a power-intensive application means many tasks and thus a high probability of having a delay. By monitoring the two applications for 480 seconds using Powerstat, we get a workload for $MSC = \{\tau_1[3, 1], \tau_2[7, 1]\}$ and $MPI = \{\tau_1[2, 1], \tau_2[3, 2], \tau_3[5, 1]\}$. MPI is considered to be a power-intensive component that has more tasks compared to MSC. We analyze their supply and demand bound functions and start the simulations at full processor speed ($s = 1$) and full resource supply $R[1, 1]$. We set a threshold for the worst-case delay for this model ($\sigma = 8$), the actual delay of this workload is $D = 10$, calculated using Equation 2.4. As shown in figure 4.11, results show that MPI (blue color line) is not schedulable, even at full processor speed, because dbf exceeds the sbf and the actual delay exceeds the worst-case tolerable delay of this task $D \quad \sigma$. While at MSC (red color line), no overload occurs, making our algorithm feasible to be applied at MPI. Therefore, according

to our proposed COD approach, the MPI task should be offloaded to the cloudlet to reduce the power consumption of the local device. Results show that offloading power-intensive tasks in MPI achieves 53% power saving (calculated by using Equation 3.2).

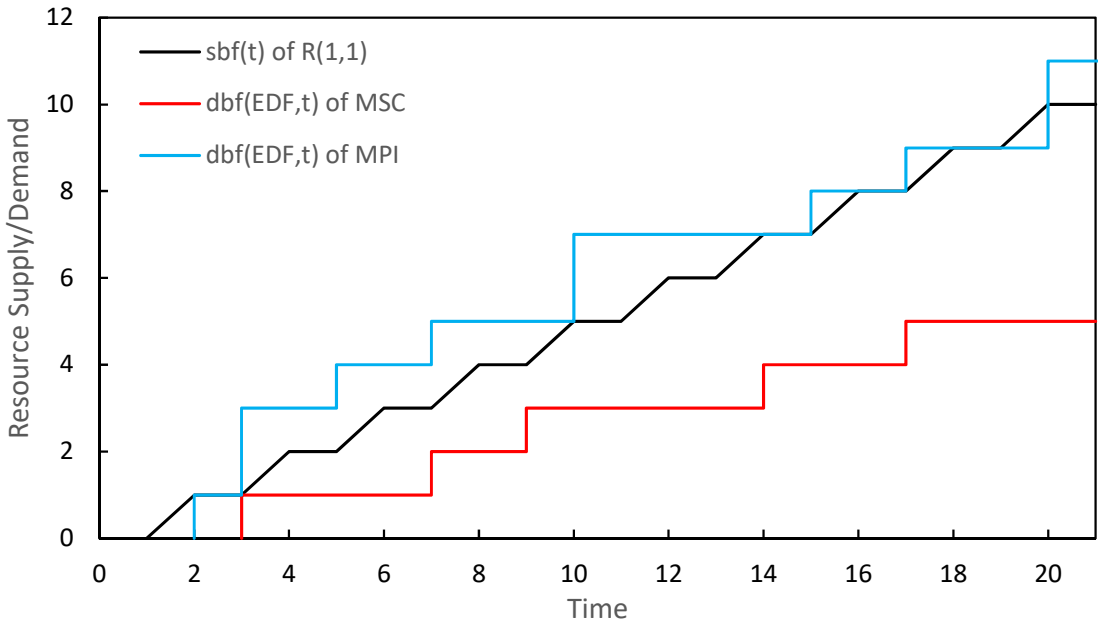


FIGURE 4.11: sbf and dbf of *MSC* and *MPI* with $R^{1}1, 1^{o}$ at full processor speed

The decision on where to offload the delay-tolerant tasks should be taken based on the system's actual delay. For example, the cloud server is more delay-tolerant than a fog (cloudlet) server, the communication delay from the fog to the user processor could be in the range of 10 ms. However, the user-to-cloud communication delay could be 100 ms; these values are taken from [51]. We should know the contact delay values of fog and cloud computing to demonstrate the feasibility of combining our proposed algorithm with fog/cloud processors. In general, the local delay (i.e., the uncertainty that we get from executing the task locally) might exceed the global delay (i.e., the delay that we obtain from processing the task to the fog/cloud processors). Therefore, we should compare the cloud/fog delay to the delay of our proposed algorithm D, and we make the offloading decision based on the number of tolerable delay values.

Table 4.7 shows the differences of cloudlet and cloud in terms of the computation delay, and the tolerable task size. Cloudlets have computation rates lying between

TABLE 4.7: Values of delay and processing capacity for cloudlets and cloud.

Processor type	Delay	Processing capacity
Fog (cloudlet)	10 ms	3000 - 5000 MIPS
Cloud	100 ms	80000 MIPS

3000-5000 Million Instructions per Seconds (MIPS). The cloud processor capacity has been taken as 80000 or more MIPS. These values are taken from previous work [51] which represent an overview of how powerful the cloud is compared to fog nodes. But the communication delay from the user to the fog processor is 10 milliseconds, which is less than the communication delay from user to cloud. Therefore, soft real-time tasks can best use cloud computing functionalities such as powerful machines, unlimited storage, processing capabilities, and reduced costs. Table 4.8 presents an assumption of the application's requirements to understand better where to offload the tasks based on the requirements. For instance, Web browsing, MSC, and Idle tasks lie in the 3000-5000 MIPS range. Therefore, they could be offloaded to the cloudlets. While MPI, chemical distillation, and inverted pendulum lie in the range of 8000 MIPS and more thus, they could be offloaded to the cloud server.

TABLE 4.8: Computation requirements assumption of existing real-time tasks.

Task	Requirement (MIPS)	Description
τ_1	2000	Web browsing
τ_2	7000	MPI
τ_3	5000	MSC
τ_4	10000	Chemical distillation
τ_5	1500	Idle
τ_6	9000	Inverted pendulum

4.3 Energy-aware scheduling optimization evaluation for HRS

We evaluated the proposed energy-aware scheduling optimization model using a set of avionics data provided by [56]. The avionics workload consists of seven components, and the details of the avionics benchmark are presented in Table 4.9. Each component was scheduled under EDF scheduling algorithm, the total energy consumption $E_t = \sum_{T_i \in W} \frac{H}{p_i}.e_i.{}^1s_i{}^{o2}$ is used at both task and component levels, and considered as a minimization objective function. No delay is acceptable for this workload to maintain the schedulability of the system. We use Matlab and AMPL/KNITRO solver to solve this optimization problem, which returns the optimum processor speed. The algorithm used in the Knitro solver is Interior-Point/Barrier Direct algorithm. As shown in Fig 4.12. for OPT-CDSS, there is a 6%, 36%, 17%, 3%, and 7% energy reduction in c1, c2, c3, c4, and c5 respectively, compared to normal existing DVS scheme (i.e., when the processor runs at its full processor speed $s = 1$). Moreover, for OPT-TDSS scheme, there is a 16%, 36%, 20%, 23%, and 28% energy reduction in c1, c2, c3, c4, and c5 respectively. This shows an average of 25% energy reduction with OPT-TDSS and 14% energy reduction with OPT-CDSS.

4.4 Economic results

In this section, we investigate the economic feasibility of using the cloud to process different processes requiring different power levels. Both computations (MSC and MPI) are performed on the laptop and are offloaded to the AWS EC2. The annual cost of performing both computations using the laptop and the cloud is calculated and compared. For the laptop, the cost is calculated by multiplying the total power consumption by the electricity cost, while for the cloud, the cost is calculated based on the AWS fee. For the laptop, the electricity price in the City of Toronto is $0.17/kWh [10], while the AWS charges an hourly fee based on the needed memory. We used a 0.5 GiB memory for the Matlab simple calculation, which costs $0.0058/hour for Linux-powered systems. In contrast, we used a 1.0 GB

TABLE 4.9: Workload specifications for a set of avionics data

Components	Workload		Resource	
	p_i	e_i	$_i$	θ_i
C1	25 50	1.4 3.9	10	2
C2	50	2.8	10	0.9
C3	50	1.4	10	0.3
C4	25 50 100 200	1.1 1.8 2 5.3	10	1.4
C5	50 200	1.3 1.5	10	0.4

memory for the Matlab power-intensive simulation, which requires $0.0116/hour for Linux-powered systems.We assumed that both the MSC and MPI run all year round (i.e., 8760 hours); this is just a monitoring period assumption to calculate the power consumption for running MSC and MPI for an entire year. Table 4.10 shows that the annual power consumption for performing the Matlab simple calculation and the Matlab power-intensive simulation is 93 kWh and 835 kWh, respectively.

Figure 4.13 shows the monthly cost for executing MSC on the computing device and the AWS EC2. Performing the MSC on the laptop is much less expensive than the AWS. For instance, the annual cost for performing MSC on the laptop and the AWS is $16 and $52, respectively. This is because the smallest available memory on the AWS is 0.5 GiB, which is too big for the MSC. Figure 4.14 shows the cumulative monthly cost for executing the Matlab power-intensive simulation on the laptop and the AWS EC2. Performing the MPI on the cloud is less expensive than the laptop. The annual cost for executing the MPI on the laptop and the AWS is $142 and $103, respectively. Therefore, the power costs for

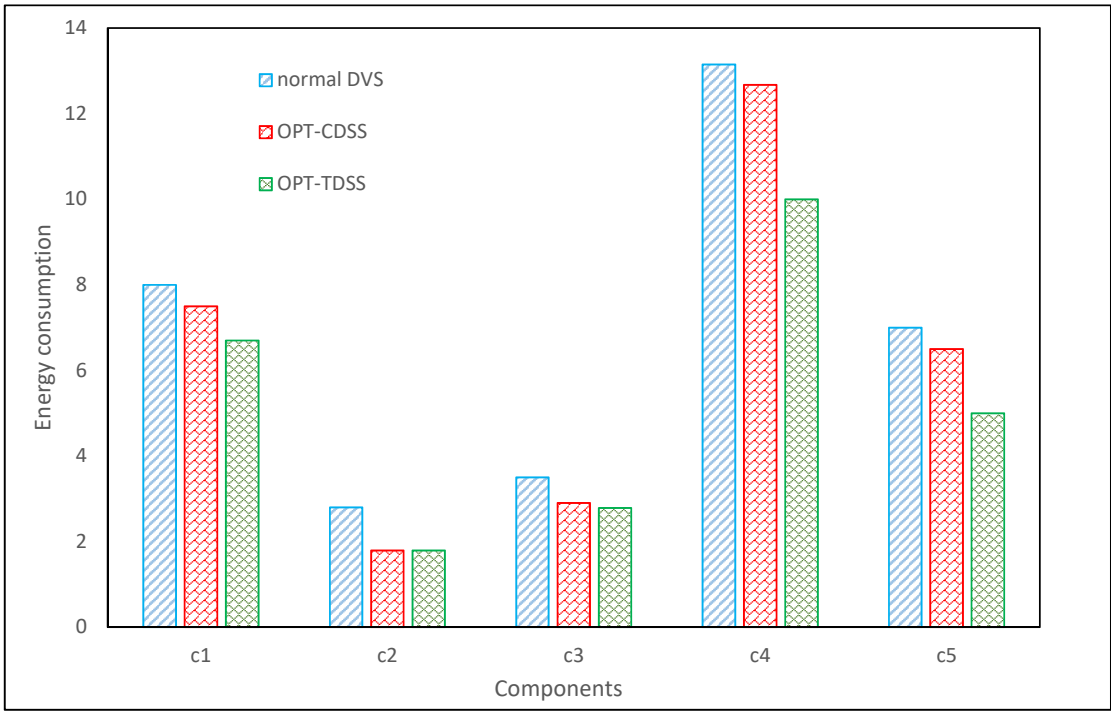


FIGURE 4.12: Energy consumption for the workload of Table 4.9

TABLE 4.10: Comparison between cumulative monthly power consumption of the laptop for the Matlab simple calculation and the matlab power intensive simulation operating conditions.

Month	MSC (kWh)	MPI (kWh)
1	7.7	69.6
2	15.4	139.1
3	23.1	208.7
4	30.8	278.3
5	46.2	347.8
6	53.9	417.4
7	61.6	486.9
8	69.3	556.5
9	77.0	629.1
10	84.7	695.6
11	88.1	765.2
12	92.4	834.8

executing the MPI can be reduced by up to 27% if executed on the cloud instead of the local computational device. It should be noted that the electricity price has a significant influence on the total cost of performing these two computations on the computing device. Therefore, the results obtained in the current study reflect the economic feasibility of using a cloud where the electricity price is $0.17/kWh.

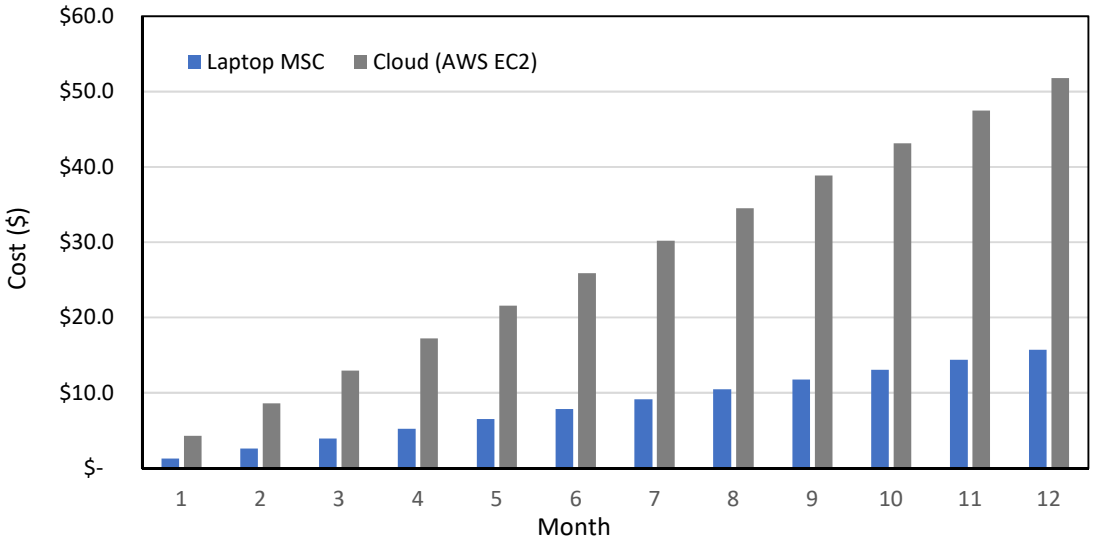


FIGURE 4.13: Comparison between the cumulative monthly cost of processing a MSC Simulation on the laptop and the cloud.

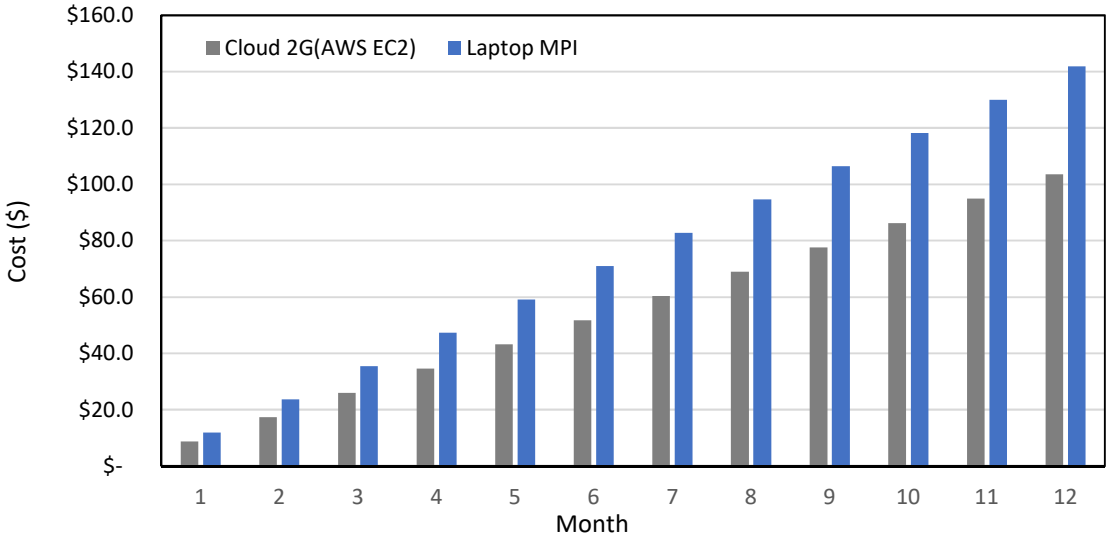


FIGURE 4.14: Comparison between the cumulative monthly cost of processing a MPI on the laptop and the cloud.

4.5 Empirical analysis applicable on other systems

To investigate the feasibility of our scheme for other systems that can tolerate a bounded overload, we used the specifications that have been explained in Example

1, with workload w = $\{\tau_1[9, 1^o], [15, 2^o]\}$. As shown in Table 4.11, the longer the worst-case delay the system can tolerate, the more energy savings can be achieved. Figure 4.15 illustrates that no overloads occur for cases 1 and 2; thus, the system consumes high energy. However, in cases 3, 4, and 5, our scheme achieves more energy savings for a system that can tolerate different worst-case bounded delays.

TABLE 4.11: Energy consumption with different *DSS* level using Example 1 specifications

Case	Processor speed level S	Worst-case tolerable delay σ	Energy consumption E_T
1	1	0	11
2	0.8	0	7.04
3	0.7	3	5.39
4	0.6	14	3.96
5	0.5	24	2.75

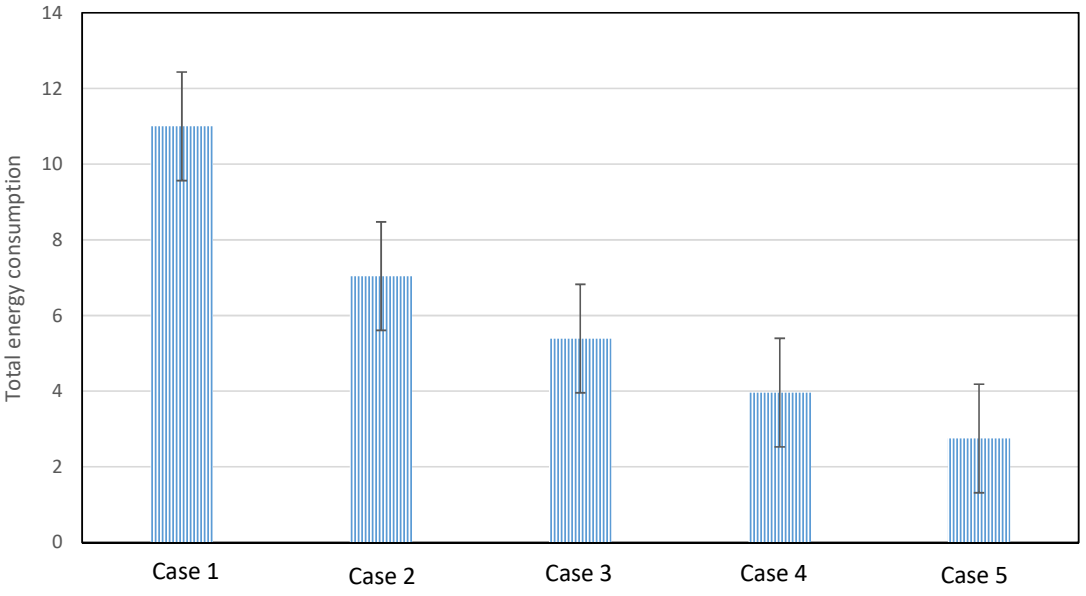


FIGURE 4.15: Energy consumption for different processor speed levels and different tolerable worst-case delay from table 4.11

To understand the task parameters that affect energy consumption, we have performed simulations over four different tasks with changes in their periods or the execution time for a processor speed level of 0.2. Table 4.12 shows the energy consumption for tasks with increasing periods for constant execution time. We observe no changes in power consumption for a gradual increase in the periods because increasing periods also increase the hyper period. On the other hand, we see an increase (Table 4.13) in energy consumption for a gradual increment in execution time with having a constant period. This is because the hyper period remains the same but the utilization of the tasks increases.

‘

TABLE 4.12: Energy consumption for tasks with increasing periods with a constant execution time

P1	E1	P2	E2	P3	E3	P4	E4	Speed level	Energy (units)
3	1	3	1	3	1	3	1	0.2	0.16
4	1	4	1	4	1	4	1	0.2	0.16
5	1	5	1	5	1	5	1	0.2	0.16
6	1	6	1	6	1	6	1	0.2	0.16
7	1	7	1	7	1	7	1	0.2	0.16
8	1	8	1	8	1	8	1	0.2	0.16
9	1	9	1	9	1	9	1	0.2	0.16
10	1	10	1	10	1	10	1	0.2	0.16

TABLE 4.13: Energy consumption for tasks with increasing execution time with a constant period

P1	E1	P2	E2	P3	E3	P4	E4	Speed level	Energy (units)
10	1	10	1	10	1	10	1	0.2	0.16
10	2	10	2	10	2	10	2	0.2	0.32
10	3	10	3	10	3	10	3	0.2	1.44
10	4	10	4	10	4	10	4	0.2	1.28
10	5	10	5	10	5	10	5	0.2	0.8
10	6	10	6	10	6	10	6	0.2	2.88
10	7	10	7	10	7	10	7	0.2	7.84
10	8	10	8	10	8	10	8	0.2	5.12
10	9	10	9	10	9	10	9	0.2	12.96

Resource provisioning is an efficient energy management technique for real-time systems. Therefore, our work can be extended to optimizing power consumption

and improving the scalability of the model. For example, a compositional structure can integrate the most acceptable resource supply for each specification and set a schedule that satisfies the workload requirements. Each resource supply is converted into a workload requirement by using the compositional framework (Figure. 4.16.).

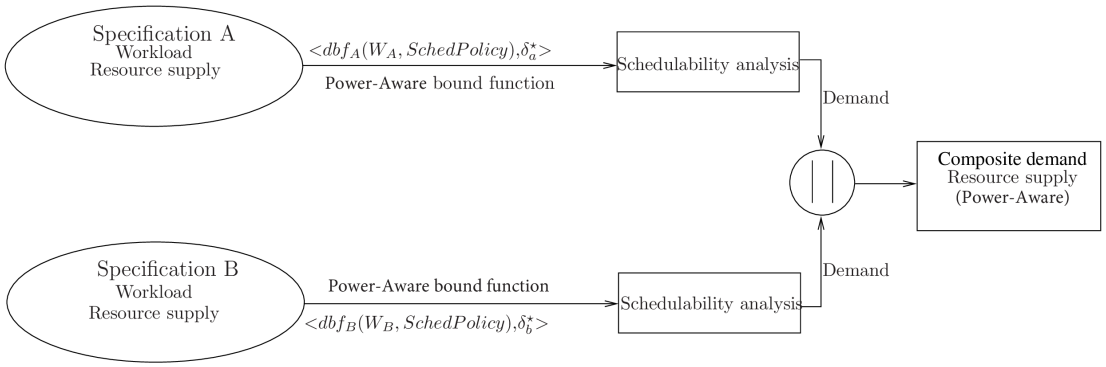


FIGURE 4.16: Compositionality analysis of overload-tolerant systems

We compute the *sbf* and *dbf* for each component to verify its workload schedulability. The compositional operator k demonstrates that multiple timing specifications can be combined into a single timing requirement. For example, the specifications we have used in our model have workload $W = \{\tau_1(2, 10), \tau_2(15, 10)\}$ and a resource supply $R(2, 10)$. Assume we want to add more tasks in the workload $W = \{\tau_i(p_i, e_i)\}$, the resource supply will be used as a workload demands under the scheduling algorithm and including any new tasks instead of dealing with all tasks together. Thus, the new workload will be $\{\tau_1(2, 10), \tau_i(p_i, e_i)\}$.

Chapter 5

Conclusion and future work

Current cyber-physical systems are complex, built with heavy over-provisioning of the resources, and consume more power than required to compromise functionality and meet the timing requirements.
Preventing the over-provisioning leads to an efficient resource reservation, but a temporary bounded delay might occur. However, some systems can tolerate bounded delays such as control systems, and thus high over-provisioning is not needed. This will then stretches the overall workload resource demand by reducing the processor speed to minimize energy consumption.
For hard real-time systems, only reducing the resources over-provisioning is an efficient technique to minimize energy consumption by adjusting the processor speed under some constraints when the resource supply cannot drop below the workload demand, thus guaranteeing the deadlines of individual tasks inside their components.

We propose an energy-efficient dynamic speed scaling (Dynamic-DSS) algorithm by permitting the workload demand to exceed the resource supply for a bounded time interval without affecting the system's schedulability for overload-tolerant systems, by selecting the appropriate processor speed level, instead of using the full resource usage. Moreover, We utilize the optimal processor speed level and get an appropriate resource supply for extra energy savings.
We propose an efficient energy-aware scheduling optimization technique based on

DSS for task and component levels (OPT-TDSS, OPT-CDSS). We use Matlab and AMPL/KNITRO solver to solve the optimization problem, which returns the optimum processor speed under some constraints that efficiently estimate the resource requirements by having the resource supply almost equal to the workload demand.
The simulation results demonstrate that our proposed scheme under the EDF scheduling algorithm achieves energy savings of up to 78.5% compared to [27]'s proposal (OPT-TSDS) that didn't consider any tolerable delay in the system.

Another challenge of cyber-physical systems is limited resources because of the growing demand for communication and computation. Due to limited resources, most complex real-time systems consume a significant amount of power and fail to satisfy the timing requirements. The automotive design is an example of real-time systems, which have a variety of tasks. In which some can tolerate a certain amount of delay. Therefore, delay-tolerant tasks (weakly hard and soft real-time tasks) can be extended to more powerful processors to be executed on cloudlet and cloud resources. We study the feasibility of offloading the delay-tolerant tasks for energy savings, and efficient offloading decision approaches are taken for weakly hard real-time tasks (based on supply and demand bound functions), and for soft real-time tasks (based on the power consumption and the processor usage of the local device).

In addition, the technical and economic feasibility of using cloud computing are experimentally investigated. The cost of executing a low-power and a power-intensive task on the local embedded processor and the cloud is calculated and compared. Results show that offloading power-intensive tasks (MPI) to the cloud achieves 53% power savings. The cost of executing the MPI task onto the cloud (AWS EC2) can reduce the power costs by up to 27%, compared to performing the same task on a local computational device. It is concluded that efficient resource reservation techniques and offloading intensive power computations onto the cloud are cost-wise and energy-efficient solutions for hard and delay-tolerant real-time

tasks, respectively.
This current study only focuses on dynamic power consumption; however, considering static power consumption will be efficient for an ideal power model. Therefore, we plan to discuss static power consumption in our future work.

Reducing energy consumption by creating efficient interfaces and resource provisioning is a promising area of research. The current work can be extended not only to optimize power consumption but also scalability. For instance, a compositional framework can combine different specifications and build a schedule that satisfies the workload demands. Moreover, we could consider GPU in the future work.

Power consumption is a summation of static power consumption due to leakage current and dynamic power consumption due to signal switching activities. Dynamic power represents the dominant component of the total power consumption because there is a quadratic relationship between the supply voltage and dynamic power consumption. This current study only focuses on dynamic power consumption; however, considering static power consumption will be efficient for an ideal power model. Moreover, running any algorithm in embedded systems consumes energy that we haven't considered in our work. Therefore, integrating static power consumption to dynamic power and evaluate the energy consumption of running our algorithm will be potentially considered in our future work.

It is essential to discuss that our proposed model is based on worst-case execution time analysis. Therefore, the real tested use-cases on the real systems do not deviate much from our analysis. Our model is highly abstracted to capture uncertainties to add any deviations from the worst-case, although the systems run mostly the average-case.
Our main energy-efficient proposed approaches are 1- Energy-efficient DSS for periodic resource model, 2- Computational offloading decision technique, and 3- Optimal energy-aware optimization technique. Each method can reduce the energy

consumption of CPS and can be used individually or combine two at offline or online (runtime) computation. For instance, we can combine the energy-efficient DSS and offloading techniques at runtime computation or the energy-aware scheduling optimization approach and offload at offline computation. The optimal energy-aware optimization technique has more overhead than the energy-efficient DSS because we need to do the analysis at offline computation before running the system.

Bibliography

[1] Suzanne Elashri and Akramul Azim. An energy-efficient periodic resource model for bounded delay-tolerant real-time systems. *IEEE Access*, 2021.

[2] Suzanne Elashri and Akramul Azim. Energy-efficient offloading of real-time tasks using cloud computing. *Cluster Computing*, pages 1–16, 2020.

[3] Andreas Berl, Erol Gelenbe, Marco Di Girolamo, Giovanni Giuliani, Hermann De Meer, Minh Quan Dang, and Kostas Pentikousis. Energy-efficient cloud computing. *The computer journal*, 53(7):1045–1051, 2010.

[4] Dave Anderson, Jim Dykes, and Erik Riedel. More than an interface-scsi vs. ata. In *FAST*, volume 2, page 3, 2003.

[5] Nasro Min-Allah, Samee U Khan, Xiuli Wang, and Albert Y Zomaya. Lowest priority first based feasibility analysis of real-time systems. *Journal of Parallel and Distributed Computing*, 73(8):1066–1075, 2013.

[6] Amjad Mahmood, Salman A Khan, Fawzi Albalooshi, and Noor Awwad. Energy-aware real-time task scheduling in multiprocessor systems using a hybrid genetic algorithm. *Electronics*, 6(2):40, 2017.

[7] Johan Nilsson. Real-time control systems with delays. 1998.

[8] Olivier Sename, Daniel Simon, and David Robert. Feedback scheduling for real-time control of systems with communication delays. In *Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA'03. IEEE Conference*, volume 2, pages 454–461. IEEE, 2003.

[9] A. Azim, S. Sundaram, and S. Fischmeister. An efficient periodic resource supply model for workloads with transient overloads. In *2013 25th Euromicro Conference on Real-Time Systems*, pages 249–258, July 2013. doi: 10.1109/ECRTS.2013.34.

[10] Xing Liu, Panwen Liu, Lun Hu, Chengming Zou, and Zhangyu Cheng. Energy-aware task scheduling with time constraint for heterogeneous cloud datacenters. *Concurrency and Computation: Practice and Experience*, 32(18): e5437, 2020.

[11] Insik Shin and Insup Lee. Compositional real-time scheduling framework with periodic model. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):30, 2008.

[12] Akramul Azim. Overloads in compositional embedded real-time control systems. In *Proceedings of the 27th International Symposium on Rapid System Prototyping: Shortening the Path from Speci cation to Prototype*, pages 51–57. ACM, 2016.

[13] AHMAD GOLCHIN. Control based tickless scheduling. 2017.

[14] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Transactions on Computers*, 53(5): 584–600, May 2004. ISSN 0018-9340. doi: 10.1109/TC.2004.1275298.

[15] Padmanabhan Pillai and Kang G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. *SIGOPS Oper. Syst. Rev.*, 35(5): 89–102, October 2001. ISSN 0163-5980. doi: 10.1145/502059.502044. URL `http://doi.acm.org/10.1145/502059.502044`.

[16] Mario Bambagini, Mauro Marinoni, Hakan Aydin, and Giorgio Buttazzo. Energy-aware scheduling for real-time systems: A survey. *ACM Transactions on Embedded Computing Systems (TECS)*, 15(1):1–34, 2016.

[17] Pengcheng Huang, Pratyush Kumar, Georgia Giannopoulou, and Lothar Thiele. Energy efficient dvfs scheduling for mixed-criticality systems. In

2014 International Conference on Embedded Software (EMSOFT), pages 1–10. IEEE, 2014.

[18] Hakan Aydin, Rami Melhem, Daniel Mossé, and Pedro Mejía-Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Transactions on Computers*, 53(5):584–600, 2004.

[19] Ghofrane Rehaiem, H Gharsellaoui, and S Ben Ahmed. Real-time scheduling approach of reconfigurable embedded systems based on neural networks with minimization of power consumption. *IFAC-PapersOnLine*, 49(12):1827–1831, 2016.

[20] Saowanee Saewong and Ragunathan Rajkumar. Practical voltage-scaling for fixed-priority rt-systems. In *Real-Time and Embedded Technology and Applications Symposium, 2003. Proceedings. The 9th IEEE*, pages 106–114. IEEE, 2003.

[21] "Yi wen Zhang and Rui feng Guo". Power-aware fixed priority scheduling for sporadic tasks in hard real-time systems. *"Journal of Systems and Software"*, "90":"128 – 137", "2014". ISSN "0164-1212".

[22] Vinay Devadas and Hakan Aydin. Real-time dynamic power management through device forbidden regions. In *2008 IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 34–44. IEEE, 2008.

[23] Youcheng Sun and Marco Di Natale. Weakly hard schedulability analysis for fixed priority scheduling of periodic real-time tasks. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):1–19, 2017.

[24] Rajib Mall. *Real-time systems: theory and practice*. Pearson Education India, 2009.

[25] Marko Bertogna. Real-time scheduling analysis for multiprocessor platforms. *Scuola Superiore Sant'Anna, Pisa*, 2008.

[26] S. Baruah, D. Chen, S. Gorinsky, and A. Mok. Generalized Multiframe Tasks. *Real-Time Systems*, 17(1):5–22, 1999.

[27] Tchamgoue, Guy Martin, Kim, Kyong Hoon, and yong Kee. Power-aware scheduling of compositional real-time frameworks. pages "58 – 71". ISSN "0164-1212".

[28] Dakai Zhu. Reliability-aware dynamic energy management in dependable embedded real-time systems. In *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, pages 397–407. IEEE, 2006.

[29] Jheng-Ming Chen, Kuochen Wang, and Ming-Ham Lin. Energy efficient scheduling for real-time systems with mixed workload. In *International Conference on Embedded and Ubiquitous Computing*, pages 33–44. Springer, 2007.

[30] Kyong Hoon Kim, Rajkumar Buyya, and Jong Kim. Power aware scheduling of bag-of-tasks applications with deadline constraints on dvs-enabled clusters. In *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid'07)*, pages 541–548. IEEE, 2007.

[31] Huaming Wu, Yi Sun, and Katinka Wolter. Energy-efficient decision making for mobile cloud offloading. *IEEE Transactions on Cloud Computing*, 8(2): 570–584, 2018.

[32] Yu-Lin Jiang, Ya-Shu Chen, Su-Wei Yang, and Chia-Hsueh Wu. Energy-efficient task offloading for time-sensitive applications in fog computing. *IEEE Systems Journal*, 13(3):2930–2941, 2018.

[33] Xue Lin, Yanzhi Wang, Qing Xie, and Massoud Pedram. Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment. *IEEE Transactions on Services Computing*, 8 (2):175–186, 2014.

[34] Liyao Xiang, Shiwen Ye, Yuan Feng, Baochun Li, and Bo Li. Ready, set, go: Coalesced offloading from mobile devices to the cloud. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 2373–2381. IEEE, 2014.

[35] Zhiyuan Li, Cheng Wang, and Rong Xu. Computation offloading to save energy on handheld devices: a partition scheme. In *Proceedings of the 2001 international conference on Compilers, architecture, and synthesis for embedded systems*, pages 238–246, 2001.

[36] Feng Xia, Fangwei Ding, Jie Li, Xiangjie Kong, Laurence T Yang, and Jianhua Ma. Phone2cloud: Exploiting computation offloading for energy saving on smartphones in mobile cloud computing. *Information Systems Frontiers*, 16 (1):95–111, 2014.

[37] Michel Goraczko, Jie Liu, Dimitrios Lymberopoulos, Slobodan Matic, Bodhi Priyantha, and Feng Zhao. Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems. In *2008 45th ACM/IEEE Design Automation Conference*, pages 191–196. IEEE, 2008.

[38] Gonzalo Huerta-Canepa and Dongman Lee. An adaptable application offloading scheme based on application behavior. In *22nd International Conference on Advanced Information Networking and Applications-Workshops (aina workshops 2008)*, pages 387–392. IEEE, 2008.

[39] Akli Abbas, Emmanuel Grolleau, Malik Loudini, and Walid-Khaled Hidouci. A real-time feedback scheduler based on control error for environmental energy harvesting systems. In *2015 International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS)*, pages 1–9. IEEE, 2015.

[40] Mehdi Kargahi and Ali Movaghar. A method for performance analysis of earliest-deadline-first scheduling policy. *The Journal of Supercomputing*, 37 (2):197–222, 2006.

[41] Yi-wen Zhang. System level fixed priority energy management algorithm for embedded real time application. *Microprocessors and Microsystems*, 64:170–177, 2019.

[42] Roberto Medina and Liliana Cucu-Grosjean. Work-in-progress: Probabilistic system-wide dvfs for real-time embedded systems. In *2019 IEEE Real-Time Systems Symposium (RTSS)*, pages 508–511. IEEE, 2019.

[43] Anantha P Chandrakasan, Samuel Sheng, and Robert W Brodersen. Low-power cmos digital design. *IEICE Transactions on Electronics*, 75(4):371–382, 1992.

[44] Keqin Li. Energy efficient scheduling of parallel tasks on multiprocessor computers. *The Journal of Supercomputing*, 60(2):223–247, 2012.

[45] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for reduced cpu energy. In *Mobile Computing*, pages 449–471. 1994.

[46] Luca Santinelli, Mauro Marinoni, Francesco Prosperi, Francesco Esposito, Gianluca Franchino, and Giorgio Buttazzo. Energy-aware packet and task co-scheduling for embedded systems. In *Proceedings of the tenth ACM international conference on Embedded software*, pages 279–288. ACM, 2010.

[47] Ravindra Jejurikar, Cristiano Pereira, and Rajesh Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proceedings of the 41st annual Design Automation Conference*, pages 275–280. ACM, 2004.

[48] Anas Toma and Jian-Jia Chen. Computation offloading for real-time systems. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1650–1651, 2013.

[49] Lee W Atkinson. Increased processor performance comparable to a desktop computer from a docked portable computer, March 16 1999. US Patent 5,884,049.

[50] Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava. A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, 18(1):129–140, 2013.

[51] Nitin Auluck, Akramul Azim, and Kaneez Fizza. Improving the schedulability of real-time tasks using fog computing. *IEEE Transactions on Services Computing*, 2019.

[52] Alexandru Iosup, Simon Ostermann, M Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Transactions on Parallel and Distributed systems*, 22(6):931–945, 2011.

[53] Richard H Byrd, Mary E Hribar, and Jorge Nocedal. An interior point algorithm for large-scale nonlinear programming. *SIAM Journal on Optimization*, 9(4):877–900, 1999.

[54] Mona A Abou-Of, Ahmed H Taha, and Amr A Sedky. Trade-off between low power and energy efficiency in benchmarking. In *Information and Communication Systems (ICICS), 2016 7th International Conference on*, pages 322–326. IEEE, 2016.

[55] Zhen Liu, Yongchao Xiang, and Xiaoya Qu. Workload-aware and cpu frequency scaling for optimal energy consumption in vm allocation. *Mathematical Problems in Engineering*, 2014, 2014.

[56] Arvind Easwaran, Insup Lee, Oleg Sokolsky, and Steve Vestal. A compositional scheduling framework for digital avionics systems. In *2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 371–380. IEEE, 2009.